山东大学 软件 学院

众智科学与网络化产业课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202000300125 | 姓名：贾星宇 | | 班级：2020级5班 |
| 实验题目：**友谊悖论验证** | | | |
| 实验学时：4 | | 实验日期：2022年4月21日 | |
| 实验目的：  1.加深对“友谊悖论”的理解：大多数人认为, 自己的朋友比自己拥有更多的朋友。  2.通过“友谊悖论”，来理解抽样偏差的一种表现形式：即一个人如果本身拥有较多朋友，那么这个人更有可能同时也是我的朋友；而一个人如果本身朋友就很少，那么这个人和我成为朋友的可能也会较小。  3.理解社会上的一些其他现象，比如学术圈内，大多数人认为，自己的合作者比自己发表了更多的论文；再如社交媒体上，大多数人认为，自己关注的人比自己拥有更多的粉丝。 | | | |
| 硬件环境：  Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz | | | |
| 软件环境：  CLion 2021.2.3 | | | |
| 实验步骤与内容：  【题目要求】  /\*  \*  输入：任意图的邻接矩阵（考察随机图和社会网络两种图，模拟生成）  输出：  1）符合友谊悖论的节点占比  相关定义：  友谊悖论：是一种社会现象, 指大多数人认为, 自己的朋友比自己拥有更多的朋友  \*/  【前提】  我们用一张无向网状结构G=（V，E）来表示社会关系：如果边（u，v）∈V，则表示点u和点v互为朋友；  【验证方法】  首先构造随机无向网状结构；  随后随机挑选一个节点a，统计它的朋友个数，并且统计其邻接的节点的朋友个数并计算其平均值x，与节点a的朋友个数相比较；  依次对每个节点进行如上方法的比较，计算出x>a节点度数的节点占比。  【核心代码】  经过本实验的撰写，发现此次实验较难的点在于如何随机生成无向网状结构，并思考出了如下代码：  前端代码：向后端服务器发送数据，包括需要随机创建的无向网状结构点的数目和边的数目：  autoCreate(){  if(this.nodeNum===''||this.edgeNum===''){  this.isNotAllFill=true;  }else if(this.edgeNum>this.nodeNum\*(this.nodeNum-1)){  this.isManyEdges=true;  }else{  this.ifCircle=true;  request.post("/api/edge/autoCreate",{  begnode:this.nodeNum,  endnode:this.edgeNum  }) .then( (response) => {  console.log(response);  this.hideAllEdges();  this.getNodes();  })  .catch((error) => {  console.log(error);  });  },  后端代码：  基本实现思想如下：  首先随机创建顶点，顶点的属性（横纵坐标）由随机数生成；  随后根据随即创建的顶点来创建边：  进入for循环，随机生成一个拥有两个顶点的边，如果此边存在，则i不变，再次进入for循环；如果此边不存在，则创建此边并且允许i++，实现随机创建。  Controller：  *//begnode:点个数；endnode:边个数* @RequestMapping(value = "/autoCreate",method = RequestMethod.*POST*) public Result<?> autoCreate(@RequestBody Edges edges){  if (nodeMapper.selectAll().length!=0) {  nodeMapper.deleteAll();  edgeMapper.deleteAll();  }  int nNum = edges.getBegnode();  int eNum = edges.getEndnode();  for (int i = 0; i < nNum; i++) {  String x = String.*valueOf*(new Random().nextInt(76) + 5)+"%";  String y = String.*valueOf*(new Random().nextInt(76) + 5)+"%";  nodeMapper.insertNode(i, x, y);  }  for (int i = 0; i < eNum;) {  int begNode = new Random().nextInt(nNum);  int endNode = new Random().nextInt(nNum);  if (begNode==endNode|| !ObjectUtils.*isEmpty*(edgeMapper.selectEdgeByNode12(begNode,endNode)))continue;  int weight = new Random().nextInt(20);  edgeMapper.addEdge(begNode, endNode, weight);  i++;  }  return Result.*success*(); }  Mapper：  @Mapper @Repository public interface NodeMapper extends BaseMapper<Nodes> { *// @Select("select \* from `user` where id = #{id}") // public User selectUserById(@Param("id")String id);* @Insert("INSERT INTO `shujvjiegou`.`nodes` (`nodeid`, `x`, `y`) VALUES (#{nodeid}, #{x}, #{y})")  public void insertNode(@Param("nodeid")int nodeid,@Param("x")String x,@Param("y")String y);   @Select("SELECT \* FROM shujvjiegou.nodes where nodeid=#{nodeid}")  public Nodes selectById(@Param("nodeid")int nodeid);   @Select("SELECT \* FROM shujvjiegou.nodes")  public Nodes[] selectAll();   @Delete("DELETE FROM `shujvjiegou`.`nodes`")  public void deleteAll();   @Delete("DELETE FROM `shujvjiegou`.`nodes` where nodeid=#{nodeid}")  public void deleteById(@Param("nodeid")int nodeid);   @Select("select count(\*) from `shujvjiegou`.`nodes`")  public int getNodeNums();  }  @Mapper @Repository public interface EdgeMapper extends BaseMapper<Edges> {  @Insert("INSERT INTO `shujvjiegou`.`edges` (`begnode`, `endnode`, `weight`) VALUES (#{begnode},#{endnode},#{weight})")  public void addEdge(@Param("begnode")int begnode,@Param("endnode")int endnode,@Param("weight")int weight);   @Select("SELECT \* FROM shujvjiegou.edges where begnode = #{begnode} and endnode = #{endnode}")  public Edges selectEdgeByNode12(@Param("begnode")int begnode, @Param("endnode")int endnode);   @Select("SELECT \* FROM shujvjiegou.edges")  public Edges[] getAllEdges();   @Delete("DELETE FROM `shujvjiegou`.`edges` WHERE (`begnode` = #{begnode} or `endnode` = #{endnode})")  public void deleteEdgesByNode(@Param("begnode")int begnode,@Param("endnode")int endnode);   @Delete("DELETE FROM `shujvjiegou`.`edges`")  public void deleteAll();   @Delete("DELETE FROM shujvjiegou.edges ORDER BY edgesId DESC LIMIT 1")  public void deleteLast();   @Select("SELECT \* FROM shujvjiegou.edges where begnode = #{begnode}")  public Edges[] selectEdgeByBegNode(@Param("begnode")int begnode);  }  本人利用先前自己开发的网页，将随机生成无向网状结构的代码进行了可视化展示。在本次实验中，总共创建了如下几个无向网状结构：  顶点数：10 边数：20：      顶点数：10 边数：30：      【统计总结】  经过计算分析，发现在顶点数>10，边数>20的情况下，某人的所有朋友的平均朋友数大于此人的朋友数的占比大约在50%~70%之间。（0.55，0.67，0.52，0.49，0.66）  虽然比例没有达到80%及以上，但是已经满足了在社交网络上“大多数人认为”的定义，同时也便说明了“友谊悖论”——大多数人认为, 自己的朋友比自己拥有更多的朋友 | | | |
| 结论分析与体会：  1.友谊悖论是一种研究社会各种关系的有效手段，他解释了为何大多数人认为，自己的合作者比自己发表了更多的论文、为何大多数人认为，自己关注的人比自己拥有更多的粉丝等等问题。  2.经过此次实验，我还学到了随机无向网状结构的创建方法，增强了自己的编程能力和随即应对能力。  3.友谊悖论也在告诉我们，我们不应该仅仅是关注于为何，而更应该主动面向社会，结交更多有意思的人、提高自己，做更有趣的自己。  4.为了增加实验的严谨程度，在此提出另一种构造随机网状结构的方法：  初始状态下所有人互不认识, 每个人随机地选择和一些人成为朋友. 注意这样的交友活动是一方发出与另一方接受, 且我们假设只要发出交友请求, 就一定会被接受. 这使得每个人发出一次请求, 就可以获得一个朋友关系. 每个人发出的请求数量是固定的, 即每个人都可以主动交固定数量的朋友. | | | |