图分析大作业项目文档

作者

算法实现

- 姓名：李仁杰
- 院系：软件学院
- 班级：软件62班
- 学号：2016013271
- 联系方式：`ShadowIterator@hotmail.com`

GUI实现、网络构建

- 姓名：洪方舟
- 院系：软件学院
- 班级：软件62班
- 学号：2016013259
- 联系方式：`hongfz16@163.com`

使用说明

- 双击`bin/GraphPro/GraphPro.exe`打开程序
- 左侧边栏选择算法
  - `Dijkstra`算法：可以在文本框中输入开始结点`id`与目标结点`id`，或者直接在图中点选；点击`slow`按钮，将会逐步画出最短路树，最终显示两点之间的最短路；点击`fast`按钮，将会直接显示两点之间最短路；按钮下方将会显示最短路径的长度；若两点间无路径相连，也会做出相应显示；
  - `Prim`算法：需要在输入框中指定开始的结点`id`；点击`slow`按钮，将会逐步显示`Prim`算法的加边过程；点击`fast`按钮，将会直接画出最小生成树；
  - `Kruskal`算法：点击`slow`按钮，将会逐步显示`Kruskal`算法加边的过程；点击fast按钮，将会直接画出最小生成树；
  - `BC（Betweenness Centrality）`介数中心度：点击后过大约`6-8s`，将会在图中着色显示每个点的介数中心度；颜色越接近暖色，表示介数中心度越大；
  - `CC（Closeness Centrality）`紧密中心度：点击后过大约`2s`，将会在图中着色显示每个点的紧密中心度；颜色越接近暖色，表示紧密中心度越大；
  - `Connectivity(1)`连通度分析(1)：指定阈值和查询点`id`，点击`initialize`，即可显示连通分量的气泡图；气泡图中半径越大，表示该连通分量中点的个数越多；点击`Query`按钮，将会查询指定点所在连通分量，并在气泡图中高亮；若需要修改阈值，在文本框中修改后点击`Redraw`即可；
  - `Connectivity(2)`连通度分析(2)：指定阈值，点击`initialize`，即可显示将所有连通分量内部点用最小生成树连接后的图；若需要修改阈值，在文本框中修改后点击`Redraw`即可；
- 控制按钮：
  - `Reset`按钮：重置按钮，仅当没有算法在工作的时候有效；
  - `Cancel`按钮：取消按钮，对于所有点击`slow`按钮后分步显示的操作，在运行过程中点击有效；

- 结点点击后将会在左侧边栏中显示结点id以及结点对应信息；
- 若左侧边栏无法显示，请将窗口放大，左右拖宽；

开发环境

GUI及网络构建开发环境

- 操作系统：`macOS High Sierra`
- 使用框架：
  - 界面：`electron` + `d3.js`
  - dll调用：`node-ffi`
- 编译器：`g++`

核心算法API开发环境

- 操作系统: `Windows 10`
- 集成开发环境：`VS2015`

算法说明

实现了图的相关算法和相应数据结构。

图存储

采用链式前向星对图进行存储

最短路径算法

采用堆优化的`dijkstra`算法，时间复杂度是$O(m+nlogm)$。

最小生成树算法

- 堆优化的`prim`算法，时间复杂度是$O(m+nlogm)$
- `kruskal`算法，时间复杂度是$O(mlogm)$

中心度

紧密中心度

c_u 定义为结点u到其它所有结点的最短路之和，采用n趟`dijkstra`算法，然后统计每个点的紧密中心度，时间复杂度是$O(nm+n^2logn)$,注意到图可能是不连通的，如果两个点不连通，规定它们之间的距离是$INF=2^{20}$

介数中心度

b_u 定义为

$$b_u=\sum_{s\neq t\in V}\sigma_{st}(u)$$

其中$\sigma_{st}(u)$是经过u的从s到t的经过u的路径条数。
给定一个s，定义

$$b_{su}=\sum_{t\in V}\sigma_{st}(u)$$

显然有

$$b_{su} = 1 + \sum_{v \in P_s(u)} b_{sv}$$

$$b_u = \sum_{s \in V} b_{su}$$

其中

$$P_s(u) = \{v \in V | d_s(v) = d_s(u) + w(u, v)\}$$

$d_s(u)$是s到u的最短路长度,$w(u, v)$是边(u, v)的长度。
即$P_s(u)$是u在所有最短路径树上的后继点集。
所以，计算介数中心度的方法是

- 枚举一个起点u，计算以u为起点的单源最短路
- 把所有点按照到u的距离由大到小排序
- 按照这个顺序依次计算b_{su}
- 枚举完所有点之后，计算b_u

动态维护连通性算法

考虑到如果每次询问都从新计算连通性比较慢，这里采用先预处理，再回答问题的办法。我用可持久化并查集来维护图的连通性。支持以下询问：

- 给定边权阈值为ST，查询整个图有多少个连通分支
- 给定边权阈值为ST,查询结点u所在的连通分支内的结点数目
- 给定边权阈值为ST,查询结点u所在的连通分支编号

其中第一个询问时间复杂度是$O(logm)$的，后两个算法的时间复杂度是$O(log^2 n + logm)$的
具体做法如下

- 预处理
    - 最开始每个结点独自在一个集合
    - 按照边权从大到小的顺序，合并两个结点所在的集合
- 当给定询问阈值ST之后，先二分查找出这个阈值的边是在前k次合并的，然后在持久化并查集内查询第k次操作之后的相应数据

可持久化并查集

可持久化并查集在$O(log^2 n)$的时间内支持以下操作

- 给定两个元素x和y，合并x和y所在的集合
- 查询第k次合并操作之后，x所在的集合编号
- 查询第k次合并操作之后，x所在集合的大小
- 将数据恢复至第k次操作之后的状态

实现方法：合并两个集合的时候采用按秩合并，并且不采用路径压缩，那么每次合并只会有一个节点的父亲节点被更改，直接用持久化数组即可。

`持久化数组`的方法：对于一个长度为n的数组，建立一棵有n个叶子节点的完全二叉树，每次修改某个叶子节点的值，那么新建从叶子到根的路径的结点，其余结点不变。每次操作都会有一个新建的根，查询第k次操作之后第x个位置的值的时候，只需要沿着第k个根走到第x个叶子节点即可。查询和修改的时间复杂度都是$O(logn)$的，修改会带来$O(logn)$的额外空间。

GUI实现说明

- 使用`node-ffi`调用核心算法的动态链接库
- 使用`d3.js`做图可视化
- 使用`electron`包装成应用程序

网络构建说明

- 以用户为结点
  - 两个用户看过相同的电影的数量超过阈值就有边；
  - 边权整数部分为两个用户看过的相同的电影数量；
  - 边权小数部分为两用户看过的相同电影的评分的近似程度，通过计算方差并归一化来处理；
- 以电影为结点
  - 两部电影有相同用户看过的数量超过阈值就有边；
  - 边权为整数，设置为两部电影看过用户中相同的用户数量；
- `txt`文件格式
  - 第一行有两个数分别表示图中结点个数`n`和边的条数`m`
  - 下面`m`行分别有三个数字分别表示边的始点，终点和边权
- `json`文件格式
  - 有两个数组`nodes`和`links`
  - `nodes`数组中元素为点对象，含有`id`，`name`，`group`属性
  - `links`数组中元素为边对象，含有`source`始点`id`，`target`终点`id`，`weight`边权属性
- 使用说明
  - 点击`bin/DataGenerate/createGraph.exe`
  - 输入`0`或者`1`来选择以用户为结点或者电影为结点
  - 输入阈值
  - 在`bin/DataGenerate/output`目录下将会生成`json`和`txt`格式的图文件

参考资料

- GUI部分html编写采用W3School的W3.CSS Template
- d3.js绘图部分参考Force-Directed Graph

功能亮点

GUI显示优化

- 使用电影为结点的数据时，消耗大量资源的是边，所以在默认状态下图中不连边，只有在需要显示最小生成树或者最短路的时候才加上边；
- 未连边的时候，为了保持所有的点显示在屏幕中央，使用指向中心的模拟力控制住所有节点；

算法优化

- 最短路和生成树算法采用堆优化时间。
- 介数中心度采用递推的方法把$O(n^3)$优化到了$O(nm + n^2 logn)$，对于稀疏图优化效果明显。
- 中心度算法采用多线程技术优化时间。
- 用持久化并查集把查询连通性的时间从$O(m + n)$优化到了$O(logm + log^2 n)$对于大型网络，优化效果明显。