**电 子 科 技 大 学**

**实 验 报 告**

1. **实验名称：**

重要节点组排序

1. **实验内容简介：**

利用C++语言编写基于度选择策略和投票策略的排序决策的重要节点排序的算法设计和实现，并分析在不同初始种子节点数量、不同SIR模型的情况下，两种策略的感染规模的对比，从而对两种策略进行比较。

1. **实验器材（设备、元器件）**

设备信息：家用笔记本电脑

操作系统：Windows 10 Educational 21H1

处理器：AMD Ryzen 5 5600U with Radeon Graphics 2.30 GHz

软件平台：CLion 2022.2.3

1. **算法说明：**

1. 重要节点及相关概念

在复杂网络的相关知识中，对节点的重要性进行衡量是一个重要问题。重要节点在抽象意义上，可以相较其他节点能够更大程度上影响网络的结构和功能。在复杂网络结构不变的时候，源节点的选择很大程度上对实际问题中的节点选择有关。对重要节点的排序方法有许多，比如度选择（度中心性）策略、基于路径和基于特征向量的排序方式。本项目中采用度选择策略和基于投票策略的排序进行排序，并且选择的图类型都为无向无权简单图。

2. 度选择策略（Degree）的排序

度选择（Degree）排序的策略基于这样一个简单的思想：节点的度越大，那么与其相邻的节点越多，对网络的影响越大，因而重要性越高。

3. 基于投票策略（VoteRank）的排序

基于投票策略的排序是对度选择策略的一种改进，通过指标从单纯的度到投票的改变，VoteRank可以更好的综合局部的信息，从而选出更加合乎需要的节点。

4. 信息传播模型

在本项目中，评判策略有效性的方法采用SIR流行病学模型来衡量。SIR模型中，每个节点处于三种状态：易感（S），感染（I）和恢复（R）。

**五、算法分析与设计**

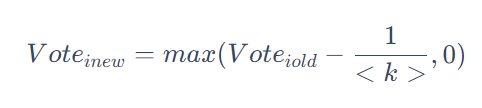
1. Degree算法部分

Degree策略选取重要节点较为简单。在读入数据集的时候，由于为无向图，可以简单的对每个节点的度进行统计，算法流程开始时，使用C++标准库中的sort函数便可在O(nlogn)的时间复杂度下对节点的度进行排序，从数据中选出度最大的k个节点作为重要节点。

2. VoteRank算法部分

VoteRank策略赋予节点两个变量：投票能力和投票分数。当初始化的时候，每个节点的投票能力都为1，并且都为相邻的节点投票。每轮的投票得分最高者便为本轮选出的重要节点。

投票并选择后，便更新各节点的投票能力。设选出的节点为U，相邻节点设为V\_i,则有以下更新规则：

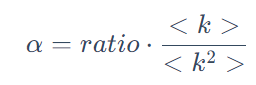


其中，<k>为图的平均度。

重复以上过程，直到选出给定规模的重要节点为止。

3. SIR模型流程

在感染流程开始前，所有节点都初始化为S状态，选取部分节点为源节点，状态置为I。每隔单位时间，每个感染节点的恢复概率为β，若已进入恢复状态便不会再改变状态，并且被感染的节点也会试图向相邻节点进行感染，感染率为α。在本项目中，为统一操作且方便观察结果，α置为设定值，例如，有以下公式[1]：



在这里，感染率α被设定为人为设定的参数ratio、图中度的期望值<k>以及度平方的期望值<k2>，通常将ratio设为1.2,视网络具体情况而定。

本项目中，β则默认为1，这也意味着节点若感染，下一单位时间就会进入R状态不变。本项目评判两种策略的有效程度的依据，主要为图中感染流程结束后处于恢复（R）状态的节点规模。

**六、实验数据及结果分析：**

1. 数据集简介

本项目的数据集主要有四份：数据集分别编号为CONT-2003[2], CONT-2005[2], Physicians[3]。

CONT-2003,CONT-2005数据集是VoteRank相关论文[1]指出认为能明显反映VoteRank和Degree策略之间的区别的数据集。其来源是科学家的论文引用过程，若引用便认为两个科学家具有合作关系。编号2003，2005表示在2003，2005年分别手机的数据集。

Physicians数据集收集了美国伊利诺斯州、Peoria，Bloomington，Quincy和Galesburg等城镇之间的246名医生的交流。每个节点代表一个医生，而两个医生之间的边表示两个医生为朋友，或者两者需要交流和讨论。

2. 项目结果与分析

本项目主要从两种情况对比重要节点组选择策略的优劣性，即在不同数量的初始种子节点，以及不同感染率α的影响下，对比感染规模的大小以得出优劣结果。鉴于各个网络节点数量、网络特征等方面差距较大，这里仅对单一的网络进行对比分析。

在以下分析过程中，N表示重要节点规模大小，ratio表示计算感染率α的人为设定参数，并已采取多次测试取平均值的方式降低概率感染带来的偶然性。

2.1 CONT-2005

在CONT-2005数据集中，相关参数：节点数量n：40421，边数m：175692。根据上述过程中的感染率计算公式，除去ratio后剩下的部分：度的平均度除图的度平方期望的值为0.0365647.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Ratio=1.2 | Ratio=1.6 | Ratio=2 | Ratio=3 | Ratio=4 | Ratio=5 |
| N=50 | 2349.95 | 4181.91 | 6216.38 | 11025.1 | 15042.6 | 18362.5 |
| N=100 | 2489.96 | 4234.21 | 6232.12 | 11015.2 | 15065.8 | 18349.8 |
| N=500 | 3223.17 | 4734.9 | 6513.35 | 11053.9 | 15068 | 18353.4 |
| N=1000 | 3935.36 | 5344.14 | 6961 | 11188.1 | 15104.4 | 18349 |
| N=3000 | 6325.89 | 7603.57 | 8944.09 | 12356.2 | 15661.5 | 18641.2 |
| N=4000 | 8306.82 | 9476.03 | 10664.5 | 13652 | 16504.6 | 19148.3 |

表1 CONT-2005 Degree策略下重要节点规模及感染率影响下的感染状况

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Ratio=1.2 | Ratio=1.6 | Ratio=2 | Ratio=3 | Ratio=4 | Ratio=5 |
| N=50 | 2373.17 | 4211.73 | 6183.62 | 11021.2 | 15061.6 | 18383.6 |
| N=100 | 2471.02 | 4246.78 | 6205.52 | 11008.51 | 15071.3 | 18359.4 |
| N=500 | 3203.18 | 4720.65 | 6520.19 | 11061.4 | 15075.2 | 18345.7 |
| N=1000 | 3961.76 | 5394.03 | 6977.59 | 11214.6 | 15099.3 | 18359.8 |
| N=3000 | 6400.25 | 7785.19 | 9247.21 | 12946.7 | 16350.2 | 19295.4 |
| N=4000 | 8535.1 | 9881.43 | 11281.5 | 14701.2 | 17835.8 | 20492.8 |

表2 CONT-2005 VoteRank策略下重要节点规模及感染率影响下的感染状况

从实验数据可以看到，当数据规模较小(N<=500)的时候，虽然已经进行了多次平均尝试降低偶然性，但过小的感染率还是对实验结果有部分影响。因此，在感染率较小时，Degree策略时较VoteRank好，时较VoteRank劣，因此可以认为在小节点规模。小感染率的情况下，两者差别较小，且容易受到小感染率的影响带来波动性。而当感染率较大时，可以看到VoteRank策略较Degree策略有了稳定的优势，但这种优势较小（10-4数量级~10-3数量级）.可以认为，由于网络特征、策略选取的某种程度上的相似性等原因，差别较小。

然而，两种策略的选取策略种，VoteRank是更加优秀的，这在CONT-2005的大数据规模下得以体现。在节点规模较大（N>500）时，感染规模开始有了明显的差距，VoteRank较Degree策略有了明显的优势（10-3~10-2数量级），最高接近10%。这说明，在合理选取感染率以及节点规模的情况下，VoteRank在某些复杂网络上的表现优于DegreeRank策略。

2.2 CONT-2003

在CONT-2003数据集中，相关参数：节点数量n：31163，边数m：120029.根据上述过程中的感染率计算公式，除去ratio后剩下的部分：度的平均度除图的度平方期望的值为0.046001.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Ratio=1.2 | Ratio=1.6 | Ratio=2 | Ratio=3 | Ratio=4 | Ratio=5 |
| N=20 | 1911.52 | 3536.42 | 5318.01 | 9456.87 | 12807.4 | 15473.8 |
| N=50 | 1997.15 | 3595.01 | 5349.11 | 9444.98 | 12823.5 | 15469.8 |
| N=100 | 2097.74 | 3612.46 | 5330.18 | 9440.38 | 12800 | 15450.2 |
| N=500 | 2846.06 | 4157.23 | 5641.93 | 9489.75 | 12814.4 | 15440.1 |
| N=1000 | 3621.94 | 4841.69 | 6176.94 | 9717.54 | 12875.6 | 15499.5 |
| N=3000 | 5885.52 | 6954.71 | 8077.3 | 10874.8 | 13519.8 | 15821 |

表3 CONT-2003 Degree策略下重要节点规模及感染率影响下的感染状况

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Ratio=1.2 | Ratio=1.6 | Ratio=2 | Ratio=3 | Ratio=4 | Ratio=5 |
| N=20 | 1909.63 | 3554.95 | 5314.74 | 9442.98 | 12798.1 | 15460.4 |
| N=50 | 1973.03 | 3565.05 | 5418.21 | 9446.74 | 12814.6 | 15472.9 |
| N=100 | 2065.8 | 3604.66 | 5328.83 | 9469.03 | 12773.1 | 15474.2 |
| N=500 | 2858.19 | 4176.61 | 5673.35 | 9530.27 | 12811.3 | 15471.4 |
| N=1000 | 3665.86 | 4902.51 | 6286.29 | 9785.03 | 12913.8 | 15539.5 |
| N=3000 | 5911.14 | 7025.57 | 8162.75 | 11012.2 | 13670.8 | 16000.7 |

表4 CONT-2003 VoteRank策略下重要节点规模及感染率影响下的感染状况

可以看到，CONT-2003数据集也符合上述结论，但实验结果并没有CONT-2005数据集显著。虽然VoteRank在较大数据规模和感染率的概率性较小的部分结果优于Degree策略，但并没有出现较大的优势，感染规模差距数量级大约为10-4~10-3.

2.3 Phisicians

在Phisicians数据集中，相关参数：节点数量n：2539，边数m：12969.根据上述过程中的感染率计算公式，除去ratio后剩下的部分：度的平均度除图的度平方期望的值为0.07569.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Ratio=1.2 | Ratio=1.6 | Ratio=2 | Ratio=3 | Ratio=4 |
| N=10 | 306.36 | 935.83 | 1389.69 | 1980.75 | 2224.38 |
| N=20 | 376.37 | 940.69 | 1398.25 | 1979.32 | 2224.74 |
| N=50 | 495.93 | 964.77 | 1411.01 | 1974.98 | 2227.2 |
| N=100 | 629.49 | 1041.38 | 1425.62 | 1979.81 | 2228.53 |
| N=200 | 781.09 | 1119.65 | 1456.11 | 1980.75 | 2224.66 |

表3 Physicians Degree策略下重要节点规模及感染率影响下的感染状况

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Ratio=1.2 | Ratio=1.6 | Ratio=2 | Ratio=3 | Ratio=4 |
| N=10 | 307.92 | 941.57 | 1407.86 | 1979.15 | 2225.19 |
| N=20 | 382.81 | 950.14 | 1388.51 | 1978.86 | 2223.76 |
| N=50 | 505.65 | 967.94 | 1408.36 | 1974.87 | 2227.46 |
| N=100 | 623.34 | 1038.9 | 1422.56 | 1984.73 | 2223.85 |
| N=200 | 786.11 | 1133.17 | 1461.7 | 1990.93 | 2232.71 |

表4 Phisicians VoteRank策略下重要节点规模及感染率影响下的感染状况

Phisicians数据集由于网络性质与上述数据集不同，作为社交网络，它的数据集没有符合Degree和VoteRank策略的最优选择，因此两种策略选出的节点差别不大，优势差距数量级小于10-4，可以忽略不计。这种情况无论是重要节点规模，抑或是感染率的变化都符合。

**七、实验分析与探讨：**

对于上述数据集，我们可以发现，对于单个的数据集，重要节点的规模对于小感染率的感染规模有重要影响，当ratio<2，即感染率大约为10-2以下的数量级时，感染规模和重要节点的规模成正比关系。而感染率越大，感染规模和重要节点的选取关系便逐渐减小。经分析可以发现，选出的节点是对复杂网络影响较大的点，因此在单一数据集的大感染率的情况下，两个重要节点选择策略都能够尽可能地使选出地节点影响力更大。

由于过小的感染率的影响，对被感染节点选择的随机性导致两种度选择策略相差不大。然而，当感染率逐渐增长，或者重要节点选择的规模逐渐扩大，随机性便被削弱，VoteRank在大多数情况下都较Degree策略有优势，少数情况由于重要节点选择、网络特征等原因，Degree策略优于VoteRank。

必须指出的一点是，Degree策略优于VoteRank策略的时候，优势并不会特别显著，而在VoteRank优于Degree的情境下，根据网络特性以及选取的参数的合适性，VoteRank可以做到大于Degree近10%的优势。因此，可以认为VoteRank策略更优于Degree策略。

**八、改进建议**

在项目过程中，遇到的最大的问题便是找不到合适的数据集以及参数。本报告选取的数据集已经是个人能够寻找到的最为典型的，能够反映Voterank和Degree策略的优劣性的数据集。实际上，作为本项目来源之一的论文[1]中，VoteRank策略的Degree策略优劣差距也不是很显著。由于时间、精力等原因，无法做到更好的结果。

其次，在编程过程中，没有更多的打磨程序过程，因此可能有性能、内存泄露、类的封装性等问题。由于项目时间问题无法进一步的改善。

另外，可以对两种算法流程进行进一步的分析，例如随机性数据所造成的方差分析、算法流程的时间开销对比等。

**九、附录**

**1. 参考论文及数据集**

[1] Jian-Xiong Zhang, Duanbing Chen, Qiang Dong, & Zhi-Dan Zhao (2016). Identifying a set of influential spreaders in complex networks Scientific Reports.

[2] M. E. J. Newman, The structure of scientific collaboration networks, *Proc. Natl. Acad. Sci. USA* **98**, 404-409 (2001).

[3] Coleman, J., Katz, E., & Menzel, H. (1957). The Diffusion of an Innovation Among Physicians*. Sociometry*, 253–270.

**2. 源代码**

**2.1 SortNetwork\_SIR头文件**

#ifndef SIR\_SORTNETWORK\_SIR\_H

#define SIR\_SORTNETWORK\_SIR\_H

#include <vector>

#include <map>

#include "string"

#include "set"

#include <cmath>

#include "random"

enum SIR\_TYPE{

SUSCEPTIBLE=1,INFECTIVE=2,REMOVAL=3

};

class SortNetwork\_SIR {

struct Node{

int id;

int SIR\_flag;

int type;

Node(){

id=0,SIR\_flag=SUSCEPTIBLE,type=0;

}

//Node(int flag,int type):SIR\_flag(flag),type(type){}

};

int totalId=0;

std::vector<std::vector<int>> Graph;

std::vector<double> Score;

std::vector<double> Vote;

std::vector<int> Node\_Degree;

std::vector<int> Degree;

std::vector<Node> SIR\_Network\_Vote,SIR\_Network\_Degree;

double averageK;

double averageK2;

std::set<int> ImportantNode\_Vote,ImportantNode\_Degree;

public:

double judgeP;

int totalNode;

int totalNR\_Vote,totalIF\_Vote,totalSU\_Vote;

int totalNR\_Degree,totalIF\_Degree,totalSU\_Degree;

SortNetwork\_SIR(const std::string& path);

void Update\_Score();

void Reset\_Network();

void Pre\_GetK();

void BeginSort(const int &NodeNum);

void PrintImportant(std::ofstream& Output\_Vote,std::ofstream & Output\_Degree);

void SIR\_Process\_Vote(std::ofstream& output,double ratio111);

void SIR\_Process\_Degree(std::ofstream& output,double ratio111);

void Print\_SIR(std::ofstream& Output\_Vote,std::ofstream & Output\_Degree);

static inline void PrintSIR\_Flag(int SIR\_TYPE,std::vector<Node>& nowNetWork,std::ofstream& fileOutput);

};

#endif //SIR\_SORTNETWORK\_SIR\_H

**2.2 SortNetWork\_SIR源文件（删去部分不重要操作）**

#include "SortNetwork\_SIR.h"

#include <fstream>

#include <iostream>

#include <sstream>

#include <algorithm>

#include <cmath>

#include <queue>

using namespace std;

SortNetwork\_SIR::SortNetwork\_SIR(const std::string &path) {

averageK=averageK2=0.0;

totalSU\_Vote= totalNode= totalIF\_Vote= totalNR\_Vote= totalId=0;

totalSU\_Degree=totalIF\_Degree=totalNR\_Degree=0;

ifstream input(path);

if(!input.is\_open()){

cout<<"Error: Can't open the data file";

exit(1);

}

string nowLine;

while(getline(input,nowLine)){

if(nowLine.empty()||nowLine[0]=='%')continue;

stringstream sinput(nowLine);

if(nowLine[0]=='#'){

char first;

int n,m;

sinput>>first>>n>>m;

totalNode=n;

Graph=vector<vector<int>>(totalNode + 1);

Score=vector<double>(totalNode + 1);

Vote=vector<double>(totalNode + 1);

SIR\_Network\_Vote=vector<Node>(totalNode + 1);

SIR\_Network\_Degree=vector<Node>(totalNode+1);

Degree=vector<int>(totalNode+1);

Node\_Degree=vector<int>(totalNode+1);

}else {

int u, v;

sinput >> u >> v;

Graph[u].push\_back(v);

Graph[v].push\_back(u);

}

}

for(int i=0; i < SIR\_Network\_Vote.size(); ++i){

SIR\_Network\_Vote[i].id=SIR\_Network\_Degree[i].id=i;

Node\_Degree[i]=i;

Vote[i]=1.0;

}

Pre\_GetK();

}

void SortNetwork\_SIR::Pre\_GetK() {

double cnt=0,cnt2=0;

auto now=Score.begin();

auto now\_Degree=Degree.begin();

for(auto &i:Graph){

\*(now\_Degree++)=int(i.size());\*(now++)=double (i.size());

cnt+=double (i.size());

cnt2+=pow(i.size(),2);

}

averageK= cnt / (double)totalNode, averageK2= cnt2 / (double)totalNode;

cout<<averageK<<' '<<averageK2<<endl;

}

void SortNetwork\_SIR::BeginSort(const int& NodeNum) {

while(ImportantNode\_Vote.size() < NodeNum){

cout<<ImportantNode\_Vote.size();

double maxs=-1.0;

int maxp=0;

for (int i = 0; i < Score.size(); ++i)

if (Score[i] > maxs&&!(ImportantNode\_Vote.count(i))) {

maxs = Score[i];

maxp = i;

}

Vote[maxp]=0.0;

for(auto&i:Graph[maxp]){

Vote[i]=max(Vote[i]-averageK,0.0);

}

//cout<<maxs<<' '<<maxp<<'\n';

ImportantNode\_Vote.insert(maxp);

//cout<<ImportantNode\_Vote.count(maxp);

Update\_Score();

}

//以下为度选择部分

sort(Node\_Degree.begin(),Node\_Degree.end(),[&](auto a,auto b){

return Degree[a]>Degree[b];

});

for(int i=0;i<Node\_Degree.size()&&i<=NodeNum;++i){

ImportantNode\_Degree.insert(Node\_Degree[i]);

}

}

void SortNetwork\_SIR::Update\_Score() {

for(int i=0; i < Score.size(); ++i){

if(ImportantNode\_Vote.count(i)){

continue;

}

Score[i]=0.0;

for(int j=0;j<Graph[i].size();++j){

Score[i]+=Vote[j];

}

}

}

void SortNetwork\_SIR::Reset\_Network() {

totalIF\_Vote=totalIF\_Degree=totalSU\_Degree=totalSU\_Vote=totalNR\_Degree=totalNR\_Vote=0;

totalId=0;

for(int i=0; i < SIR\_Network\_Vote.size(); ++i){

SIR\_Network\_Vote[i].SIR\_flag=SUSCEPTIBLE,SIR\_Network\_Vote[i].type=0;

SIR\_Network\_Degree[i].SIR\_flag=SUSCEPTIBLE,SIR\_Network\_Degree[i].type=0;

}

}

void SortNetwork\_SIR::SIR\_Process\_Vote(ofstream& output,double ratio111) {

std::random\_device rd; //linux涓嬶紝璇诲彇/dev/random鑾峰彇纭欢浜х敓鐨勯殢鏈烘暟

std::mt19937 e{rd()}; // or std::default\_random\_engine e{rd()}; 鐢℉RNG浣滀负PRNG鐨勭瀛?

std::uniform\_real\_distribution<double> uniformReal(0,1);

//cout<<"1";

totalSU\_Vote=totalNode;

judgeP=ratio111\*averageK/averageK2;

deque<int> q,temq;

vector<bool> vis(totalNode + 1, false);

for(auto i:ImportantNode\_Vote) {

q.push\_back(i);

vis[i] = true;

++totalIF\_Vote;

--totalSU\_Vote;

SIR\_Network\_Vote[i].SIR\_flag = INFECTIVE, SIR\_Network\_Vote[i].type = ++totalId;

//SIR\_Network\_Vote[i]=Node(INFECTIVE,++totalId);

}

//judgeP=2\*averageK/averageK2;

//cout<<judgeP<<endl;

int index=0;

while(!q.empty()){

//cout<<q.size()<<endl;

swap(temq,q);

q.clear();

int cnt=0;

while(!temq.empty()){

auto nowNode\_id=temq.front();

temq.pop\_front();

for(auto v:Graph[nowNode\_id]){

if(!vis[v]){

auto temP=uniformReal(e);

if(temP<judgeP){

++cnt;

q.push\_back(v);

vis[v]= true;

SIR\_Network\_Vote[v].type=SIR\_Network\_Vote[nowNode\_id].type;

SIR\_Network\_Vote[v].SIR\_flag=INFECTIVE;

++totalIF\_Vote;

--totalSU\_Vote;

}

}

}

SIR\_Network\_Vote[nowNode\_id].SIR\_flag=REMOVAL;

++totalNR\_Vote;

--totalIF\_Vote;

}

//output<<++index<<" step:\t"<<cnt<<" infected node\n";

}

}

void SortNetwork\_SIR::SIR\_Process\_Degree(ofstream& output,double ratio111) {

std::random\_device rd;

std::mt19937 e{rd()}; // or std::default\_random\_engine e{rd()}; 鐢℉RNG浣滀负PRNG鐨勭瀛?

std::uniform\_real\_distribution<double> uniformReal(0,1);

//cout<<"1";

judgeP=ratio111\*averageK/averageK2;

totalSU\_Degree=totalNode;

deque<int> q,temq;

totalId=0;

vector<bool> vis(totalNode + 1, false);

for(auto i:ImportantNode\_Degree) {

q.push\_back(i);

vis[i] = true;

++totalIF\_Degree;

--totalSU\_Degree;

SIR\_Network\_Degree[i].SIR\_flag = INFECTIVE, SIR\_Network\_Degree[i].type = ++totalId;

}

//cout<<judgeP<<endl;

int index=0;

while(!q.empty()){

//cout<<q.size()<<endl;

swap(temq,q);

q.clear();

int cnt=0;

while(!temq.empty()){

auto nowNode\_id=temq.front();

temq.pop\_front();

for(auto v:Graph[nowNode\_id]){

if(!vis[v]){

auto temP=uniformReal(e);

if(temP<judgeP){

++cnt;

q.push\_back(v);

vis[v]= true;

SIR\_Network\_Degree[v].type=SIR\_Network\_Degree[nowNode\_id].type;

SIR\_Network\_Degree[v].SIR\_flag=INFECTIVE;

++totalIF\_Degree;

--totalSU\_Degree;

}

}

}

SIR\_Network\_Degree[nowNode\_id].SIR\_flag=REMOVAL;

++totalNR\_Degree;

--totalIF\_Degree;

}

//output<<++index<<" step:\t"<<cnt<<" infected node\n";

}

}

2.3 主程序（main.cpp）

#include "SortNetwork\_SIR.h"

#include <fstream>

#include <iostream>

#include <chrono>

using namespace std;

string path=R"(D:\Desktop\_total\Class\First\_semester\_of\_sophomore\_year\Data\_Structure\_and\_Algorithm\homework\SIR\)";

string fileName="health.txt";

int NodeNum=10000;

//double voteRemoval=0,voteInfect=0,voteSus=0;

//double degreeRemoval=0,degreeInfect=0,degreeSus=0;

int totalNode=0;

int totalCnt=100;

vector<double> ratioTotal;

double total\_voteRemoval,total\_voteInfect,total\_voteSus,total\_degreeRemoval,total\_degreeInfect,total\_degreeSus;

double vari\_voteRemoval,vari\_voteInfect,vari\_voteSus,vari\_degreeRemoval,vari\_degreeInfect,vari\_degreeSus;

vector<double> voteRemoval,voteInfect,voteSus,degreeRemoval,degreeInfect,degreeSus;

ofstream DegreeOut(fileName+"\_DegreeRes.txt"),VoteOut(fileName+"\_VoteRes.txt");

double calVariance(vector<double>& total,double average,double num){

double sum=0;

for(auto &i:total){

sum+=pow(i-average,2);

}

return sum/(num-1);

}

void calculateTotal(SortNetwork\_SIR& sortNetworkSir){

totalNode=sortNetworkSir.totalNode;

//sortNetworkSir.PrintImportant(VoteOut,DegreeOut);

cout<<"step 1:vote\n";

for(int j=0;j<ratioTotal.size();++j){

VoteOut<<"The "<<j+1<<" alpha case:\n";

auto start=std::chrono::steady\_clock::now();

for(int i=0; i < totalCnt; ++i){

sortNetworkSir.SIR\_Process\_Vote(VoteOut,ratioTotal[j]);

voteRemoval.push\_back(sortNetworkSir.totalNR\_Vote);

voteInfect.push\_back(sortNetworkSir.totalIF\_Vote);

voteSus.push\_back(sortNetworkSir.totalSU\_Vote);

sortNetworkSir.Reset\_Network();

}

VoteOut<<"alpha:\t"<<sortNetworkSir.judgeP<<'\n';

auto end=std::chrono::steady\_clock::now();

std::chrono::duration<double> elapsed\_seconds=end-start;

VoteOut<<"elapsed time: "<<elapsed\_seconds.count()<<"s\n";

VoteOut<<"alpha:"<<sortNetworkSir.judgeP<<'\n';

total\_voteInfect= accumulate(voteInfect.begin(),voteInfect.end(),0);

total\_voteRemoval= accumulate(voteRemoval.begin(), voteRemoval.end(),0);

total\_voteSus= accumulate(voteSus.begin(), voteSus.end(),0);

vari\_voteInfect= calVariance(voteInfect,total\_voteInfect/totalCnt,totalCnt);

vari\_voteRemoval= calVariance(voteRemoval,total\_voteRemoval/totalCnt,totalCnt);

vari\_voteSus= calVariance(voteSus, total\_voteSus / totalCnt, totalCnt);

VoteOut << totalCnt << " round:\nthe average removal node is " <<total\_voteRemoval/(double)totalCnt << "/" << totalNode <<

", variance is "<<vari\_voteRemoval<< '\n';

VoteOut << "the average infected node is " << total\_voteInfect / (double)totalCnt << "/" << totalNode <<

", variance is "<<vari\_voteInfect<< '\n';

VoteOut << "the average susceptible node is " << total\_voteSus / (double)totalCnt << "/" << totalNode <<

", variance is "<<vari\_voteSus<<'\n';

voteSus.clear();voteInfect.clear();voteRemoval.clear();

}

cout<<"step 2: degree:";

for(int j=0;j<ratioTotal.size();++j){

DegreeOut<<"The "<<j+1<<" alpha case:\n";

auto start=std::chrono::steady\_clock::now();

for(int i=0;i<totalCnt;++i){

sortNetworkSir.SIR\_Process\_Degree(DegreeOut,ratioTotal[j]);

degreeRemoval.push\_back(sortNetworkSir.totalNR\_Degree);

degreeInfect.push\_back(sortNetworkSir.totalIF\_Degree);

degreeSus.push\_back(sortNetworkSir.totalSU\_Degree);

sortNetworkSir.Reset\_Network();

}

DegreeOut<<"alpha:\t"<<sortNetworkSir.judgeP<<'\n';

auto end=std::chrono::steady\_clock::now();

std::chrono::duration<double> elapsed\_seconds=end-start;

DegreeOut<<"elapsed time: "<<elapsed\_seconds.count()<<"s\n";

DegreeOut<<"alpha:"<<sortNetworkSir.judgeP<<'\n';

total\_degreeInfect= accumulate(degreeInfect.begin(),degreeInfect.end(),0);

total\_degreeRemoval= accumulate(degreeRemoval.begin(),degreeRemoval.end(),0);

total\_degreeSus= accumulate(degreeSus.begin(),degreeSus.end(),0);

vari\_degreeInfect= calVariance(degreeInfect,total\_degreeInfect/totalCnt,totalCnt);

vari\_degreeRemoval= calVariance(degreeRemoval,total\_degreeRemoval/totalCnt,totalCnt);

vari\_degreeSus=calVariance(degreeSus,total\_degreeSus/totalCnt,totalCnt);

DegreeOut << totalCnt << " round:\nthe average removal node is " << total\_degreeRemoval / (double)totalCnt << "/" << totalNode <<

", variance is "<<vari\_degreeRemoval<<'\n';

DegreeOut << "the average infected node is " << total\_degreeInfect / (double)totalCnt << "/" << totalNode <<

", variance is "<<vari\_degreeInfect<<'\n';

DegreeOut << "the average susceptible node is " << total\_degreeSus / (double)totalCnt << "/" << totalNode <<

", variance is "<<vari\_degreeSus<<'\n';

degreeSus.clear();degreeRemoval.clear();degreeInfect.clear();

}

}

int main() {

int n;

cin>>NodeNum>>n;

for(int i=0;i<n;i++){

double tem;

cin>>tem;

ratioTotal.push\_back(tem);

cout<<tem<<"\n";

}

SortNetwork\_SIR sortNetworkSir(path+fileName);

//cin>>fileName;

//NodeNum=sortNetworkSir.totalNode/100;

//cout<<NodeNum<<'\n';

sortNetworkSir.BeginSort(NodeNum);

//sortNetworkSir.PrintImportant(VoteOut,DegreeOut);

calculateTotal(sortNetworkSir);

//sortNetworkSir.Print\_SIR(VoteOut,DegreeOut);

cout<<"over";

return 0;

}