# 大数据-Pagerank-实验报告

潘巧巧1613415 李佳骏1613368

## 概述

语言：C++。  
分块：4×4，共16块。  
大致过程：读取原始数据wikidata-分块-获取所有网页-初始化PageRank-Map\_Reduce执行至收敛-写入数据

## 数据集分析

* 1. 原始数据集wikidata以“[FromNodeID] [ToNodeID]”形式存储，包括了有入度有出度、有入度无出度、有出度无入度等各个类型的结点。
  2. 涉及源网页6110个，所有网页点共7115个。
  3. 数据集较稀疏，结点间联系不太紧密，可知用矩阵存储空间消耗较大。

## 程序设计大致流程及关键代码解析

**1、为【优化稀疏矩阵】，本实验中数据存储使用紧凑表示法（以WebData为类型vector一维数组Data）存储数据。**

struct WebData

{

int WebID; //存储源网页号码

int NumOfOut; //存储源网页的出度

vector<int> Target; //存储源网页指向的各个目标网页号码

};

**2、Load()函数将wikidata.txt的数据读入以上数组Data中**

void load()

{

//打开wikidata.txt

ifstream infile1;

infile1.open("wikidata.txt");

assert(infile1.is\_open()); //若失败,则输出错误消息,并终止程序运行

/\*————向Data数组内存数据————\*/

string s; //存放读取行数据

string blank = " ";

int Web = -1;

vector<int> TemporaryTarget;

while (1)//逐行读取数据

{

getline(infile1, s);

if (s == "") //已经跑到最后了，要把最后一个数据存入

{

WebData save;

save.WebID = Web;

save.NumOfOut = TemporaryTarget.size();

save.Target = TemporaryTarget;

Data.push\_back(save);

break;

}

int LocBlack = s.find(blank); //空格所在位置

//获取当前源网页

string NowWeb = s.substr(0, LocBlack);

int Now\_Web = stoi(NowWeb);

//如果当前源网页和上一个网页号码不一样，则是新的网页

if (Now\_Web != Web)

{

//如果现在Data数组不是空的，对上一个源网页的收尾操作，存入Data中

if (Web != -1)

{

WebData save;

save.WebID = Web;

save.NumOfOut = TemporaryTarget.size();

save.Target = TemporaryTarget;

Data.push\_back(save);

}

//对相关数据进行重新初始化

TemporaryTarget.clear();

Web = Now\_Web; //更新当前源网页

//向临时目标数组中存入第一个数据

int leng = s.length();

string Tar = s.substr(LocBlack + 1, leng - LocBlack);

int Targ = stoi(Tar); //第一个数据

TemporaryTarget.push\_back(Targ); //存入

continue;

}

//如果当前源网页和上一个网页号码一样，则继续对上一个网页的操作

if (Now\_Web == Web)

{

int leng = s.length();

string Tar = s.substr(LocBlack + 1, leng - LocBlack);

int Targ = stoi(Tar); //获取目标网页的号码

TemporaryTarget.push\_back(Targ); //存入

continue;

}

}

infile1.close();

}

**3、为方便分块操作，对Data中的数据按照源网页号码升序排列**  
bool cmp2(WebData a, WebData b)

{

return a.WebID < b.WebID;

}  
main函数中：sort(Data.begin(), Data.end(), cmp2);

**4、【分块操作】。**

**考虑到数据量较大，本实验将数组Data中的数据分成4×4=16块，存入**vector<WebData> Map[4][4];**中。具体分组信息如下注释。**  
void fourXfour\_Map()

{

//总网页数=6110(0-6109),把他们分成4份

/\* 分组：以下数字均为Data下标，不是源网页码

A：0-1526 （1527个数据）

B：1527-3053 （1527个）

C：3054-4580 （1527个）

D：4581-6109 （1529个）

【4x4】 Map

A B C D

A 00 01 02 03

B 10 11 12 13

C 20 21 22 23

D 30 31 32 33

\*/

/\*——————AA，存入Map[0][0]——————\*/

int start1 = 0, end1 = 1526;

int start2 = 0, end2 = 1526;

int StartID2 = Data[start2].WebID;

int EndID2 = Data[end2].WebID;

for (int i = start1; i <= end1; i++)

{

//临时变量save

WebData save;

save.WebID = Data[i].WebID;

save.NumOfOut = Data[i].NumOfOut;

for (int j = 0; j < Data[i].NumOfOut; j++)

{

if (Data[i].Target[j] >= StartID2 && Data[i].Target[j] <= EndID2)

save.Target.push\_back(Data[i].Target[j]);

}

if (save.Target.size() != 0)

Map[0][0].push\_back(save);

}

/\*————————————————————\*/

/\*——————AB，存入Map[0][1]——————\*/

start1 = 0, end1 = 1526;

start2 = 1527, end2 = 3053;

StartID2 = Data[start2].WebID;

EndID2 = Data[end2].WebID;

for (int i = start1; i <= end1; i++)

{

//临时变量save

WebData save;

save.WebID = Data[i].WebID;

save.NumOfOut = Data[i].NumOfOut;

for (int j = 0; j < Data[i].NumOfOut; j++)

{

if (Data[i].Target[j] >= StartID2 && Data[i].Target[j] <= EndID2)

save.Target.push\_back(Data[i].Target[j]);

}

if (save.Target.size() != 0)

Map[0][1].push\_back(save);

}

/\*————————————————————\*/

/\*——————AC，存入Map[0][2]——————\*/

start1 = 0, end1 = 1526;

start2 = 3054, end2 = 4580;

StartID2 = Data[start2].WebID;

EndID2 = Data[end2].WebID;

for (int i = start1; i <= end1; i++)

{

//临时变量save

WebData save;

save.WebID = Data[i].WebID;

save.NumOfOut = Data[i].NumOfOut;

for (int j = 0; j < Data[i].NumOfOut; j++)

{

if (Data[i].Target[j] >= StartID2 && Data[i].Target[j] <= EndID2)

save.Target.push\_back(Data[i].Target[j]);

}

if (save.Target.size() != 0)

Map[0][2].push\_back(save);

}

/\*————————————————————\*/

/\*——————AD，存入Map[0][3]——————\*/

start1 = 0, end1 = 1526;

start2 = 4581, end2 = 6109;

StartID2 = Data[start2].WebID;

EndID2 = Data[end2].WebID;

for (int i = start1; i <= end1; i++)

{

//临时变量save

WebData save;

save.WebID = Data[i].WebID;

save.NumOfOut = Data[i].NumOfOut;

for (int j = 0; j < Data[i].NumOfOut; j++)

{

if (Data[i].Target[j] >= StartID2 && Data[i].Target[j] <= EndID2)

save.Target.push\_back(Data[i].Target[j]);

}

if (save.Target.size() != 0)

Map[0][3].push\_back(save);

}

/\*————————————————————\*/

/\*——————BA，存入Map[1][0]——————\*/

start1 = 1527, end1 = 3053;

start2 = 0, end2 = 1526;

StartID2 = Data[start2].WebID;

EndID2 = Data[end2].WebID;

for (int i = start1; i <= end1; i++)

{

//临时变量save

WebData save;

save.WebID = Data[i].WebID;

save.NumOfOut = Data[i].NumOfOut;

for (int j = 0; j < Data[i].NumOfOut; j++)

{

if (Data[i].Target[j] >= StartID2 && Data[i].Target[j] <= EndID2)

save.Target.push\_back(Data[i].Target[j]);

}

if (save.Target.size() != 0)

Map[1][0].push\_back(save);

}

/\*————————————————————\*/

/\*——————BB，存入Map[1][1]——————\*/

start1 = 1527, end1 = 3053;

start2 = 1527, end2 = 3053;

StartID2 = Data[start2].WebID;

EndID2 = Data[end2].WebID;

for (int i = start1; i <= end1; i++)

{

//临时变量save

WebData save;

save.WebID = Data[i].WebID;

save.NumOfOut = Data[i].NumOfOut;

for (int j = 0; j < Data[i].NumOfOut; j++)

{

if (Data[i].Target[j] >= StartID2 && Data[i].Target[j] <= EndID2)

save.Target.push\_back(Data[i].Target[j]);

}

if (save.Target.size() != 0)

Map[1][1].push\_back(save);

}

/\*————————————————————\*/

/\*——————BC，存入Map[1][2]——————\*/

start1 = 1527, end1 = 3053;

start2 = 3054, end2 = 4580;

StartID2 = Data[start2].WebID;

EndID2 = Data[end2].WebID;

for (int i = start1; i <= end1; i++)

{

//临时变量save

WebData save;

save.WebID = Data[i].WebID;

save.NumOfOut = Data[i].NumOfOut;

for (int j = 0; j < Data[i].NumOfOut; j++)

{

if (Data[i].Target[j] >= StartID2 && Data[i].Target[j] <= EndID2)

save.Target.push\_back(Data[i].Target[j]);

}

if (save.Target.size() != 0)

Map[1][2].push\_back(save);

}

/\*————————————————————\*/

/\*——————BD，存入Map[1][3]——————\*/

start1 = 1527, end1 = 3053;

start2 = 4581, end2 = 6109;

StartID2 = Data[start2].WebID;

EndID2 = Data[end2].WebID;

for (int i = start1; i <= end1; i++)

{

//临时变量save

WebData save;

save.WebID = Data[i].WebID;

save.NumOfOut = Data[i].NumOfOut;

for (int j = 0; j < Data[i].NumOfOut; j++)

{

if (Data[i].Target[j] >= StartID2 && Data[i].Target[j] <= EndID2)

save.Target.push\_back(Data[i].Target[j]);

}

if (save.Target.size() != 0)

Map[1][3].push\_back(save);

}

/\*————————————————————\*/

/\*——————CA，存入Map[2][0]——————\*/

start1 = 3054, end1 = 4580;

start2 = 0, end2 = 1526;

StartID2 = Data[start2].WebID;

EndID2 = Data[end2].WebID;

for (int i = start1; i <= end1; i++)

{

//临时变量save

WebData save;

save.WebID = Data[i].WebID;

save.NumOfOut = Data[i].NumOfOut;

for (int j = 0; j < Data[i].NumOfOut; j++)

{

if (Data[i].Target[j] >= StartID2 && Data[i].Target[j] <= EndID2)

save.Target.push\_back(Data[i].Target[j]);

}

if (save.Target.size() != 0)

Map[2][0].push\_back(save);

}

/\*————————————————————\*/

/\*——————CB，存入Map[2][1]——————\*/

start1 = 3054, end1 = 4580;

start2 = 1527, end2 = 3053;

StartID2 = Data[start2].WebID;

EndID2 = Data[end2].WebID;

for (int i = start1; i <= end1; i++)

{

//临时变量save

WebData save;

save.WebID = Data[i].WebID;

save.NumOfOut = Data[i].NumOfOut;

for (int j = 0; j < Data[i].NumOfOut; j++)

{

if (Data[i].Target[j] >= StartID2 && Data[i].Target[j] <= EndID2)

save.Target.push\_back(Data[i].Target[j]);

}

if (save.Target.size() != 0)

Map[2][1].push\_back(save);

}

/\*————————————————————\*/

/\*——————CC，存入Map[2][2]——————\*/

start1 = 3054, end1 = 4580;

start2 = 3054, end2 = 4580;

StartID2 = Data[start2].WebID;

EndID2 = Data[end2].WebID;

for (int i = start1; i <= end1; i++)

{

//临时变量save

WebData save;

save.WebID = Data[i].WebID;

save.NumOfOut = Data[i].NumOfOut;

for (int j = 0; j < Data[i].NumOfOut; j++)

{

if (Data[i].Target[j] >= StartID2 && Data[i].Target[j] <= EndID2)

save.Target.push\_back(Data[i].Target[j]);

}

if (save.Target.size() != 0)

Map[2][2].push\_back(save);

}

/\*————————————————————\*/

/\*——————CD，存入Map[2][3]——————\*/

start1 = 3054, end1 = 4580;

start2 = 4581, end2 = 6109;

StartID2 = Data[start2].WebID;

EndID2 = Data[end2].WebID;

for (int i = start1; i <= end1; i++)

{

//临时变量save

WebData save;

save.WebID = Data[i].WebID;

save.NumOfOut = Data[i].NumOfOut;

for (int j = 0; j < Data[i].NumOfOut; j++)

{

if (Data[i].Target[j] >= StartID2 && Data[i].Target[j] <= EndID2)

save.Target.push\_back(Data[i].Target[j]);

}

if (save.Target.size() != 0)

Map[2][3].push\_back(save);

}

/\*————————————————————\*/

/\*——————DA，存入Map[3][0]——————\*/

start1 = 4581, end1 = 6109;

start2 = 0, end2 = 1526;

StartID2 = Data[start2].WebID;

EndID2 = Data[end2].WebID;

for (int i = start1; i <= end1; i++)

{

//临时变量save

WebData save;

save.WebID = Data[i].WebID;

save.NumOfOut = Data[i].NumOfOut;

for (int j = 0; j < Data[i].NumOfOut; j++)

{

if (Data[i].Target[j] >= StartID2 && Data[i].Target[j] <= EndID2)

save.Target.push\_back(Data[i].Target[j]);

}

if (save.Target.size() != 0)

Map[3][0].push\_back(save);

}

/\*————————————————————\*/

/\*——————DB，存入Map[3][1]——————\*/

start1 = 4581, end1 = 6109;

start2 = 1527, end2 = 3053;

StartID2 = Data[start2].WebID;

EndID2 = Data[end2].WebID;

for (int i = start1; i <= end1; i++)

{

//临时变量save

WebData save;

save.WebID = Data[i].WebID;

save.NumOfOut = Data[i].NumOfOut;

for (int j = 0; j < Data[i].NumOfOut; j++)

{

if (Data[i].Target[j] >= StartID2 && Data[i].Target[j] <= EndID2)

save.Target.push\_back(Data[i].Target[j]);

}

if (save.Target.size() != 0)

Map[3][1].push\_back(save);

}

/\*————————————————————\*/

/\*——————DC，存入Map[3][2]——————\*/

start1 = 4581, end1 = 6109;

start2 = 3054, end2 = 4580;

StartID2 = Data[start2].WebID;

EndID2 = Data[end2].WebID;

for (int i = start1; i <= end1; i++)

{

//临时变量save

WebData save;

save.WebID = Data[i].WebID;

save.NumOfOut = Data[i].NumOfOut;

for (int j = 0; j < Data[i].NumOfOut; j++)

{

if (Data[i].Target[j] >= StartID2 && Data[i].Target[j] <= EndID2)

save.Target.push\_back(Data[i].Target[j]);

}

if (save.Target.size() != 0)

Map[3][2].push\_back(save);

}

/\*————————————————————\*/

/\*——————DD，存入Map[3][3]——————\*/

start1 = 4581, end1 = 6109;

start2 = 4581, end2 = 6109;

StartID2 = Data[start2].WebID;

EndID2 = Data[end2].WebID;

for (int i = start1; i <= end1; i++)

{

//临时变量save

WebData save;

save.WebID = Data[i].WebID;

save.NumOfOut = Data[i].NumOfOut;

for (int j = 0; j < Data[i].NumOfOut; j++)

{

if (Data[i].Target[j] >= StartID2 && Data[i].Target[j] <= EndID2)

save.Target.push\_back(Data[i].Target[j]);

}

if (save.Target.size() != 0)

Map[3][3].push\_back(save);

}

/\*————————————————————\*/

}  
**5、由于需要计算所有网页的pagerank值，我们需要获取所有网页的并集。  
 在读取数据并存入Data数组时，我们获取了作为源网页的所有结点。现在我们通过**vector<int> AllTarget()**函数获取所有当过目标网页点的网页的并集，再通过函数**vector<int> AllWeb(vector<int> AllTarget)**把源网页集合与目标网页集合取并集，则得到所有网页的结点**

//把所有当过目标的网页集中表示

vector<int> AllTarget()

{

vector<int> Mrow;

for (int i = 0; i < Data.size(); i++) //外层遍历所有源网页

{

for (int j = 0; j < Data[i].Target.size(); j++) //内层遍历当前源网页的目标网页

{

int now = Data[i].Target[j];

int nPosition = BinSearch(Mrow,Mrow.size(),now);

if (nPosition == -1) //当前目标网页在Mrow中不存在，则存入

{

Mrow.push\_back(now);

sort(Mrow.begin(), Mrow.end(), less<int>());

}

}

}

return Mrow;

}

//求所有源网页、目标网页的并集

vector<int> AllWeb(vector<int> AllTarget)

{

//先把所有目标网页赋给AllWebsite

vector<int> AllWebsite = AllTarget;

//遍历所有源网页，如果有是源网页但是不是目标网页的，插入

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

{

int now = Data[i].WebID;

int nPosition = BinSearch(AllWebsite, AllWebsite.size(), now);

if (nPosition == -1) //当前目标网页在Mrow中不存在，则存入

{

AllWebsite.push\_back(now);

sort(AllWebsite.begin(), AllWebsite.end(), less<int>());

}

}

return AllWebsite;

}

**6、初始化PageRank。**

**PageRank数组的大小应该是所有网页的并集的大小，PageRank中所有网页的初始分数应该为1/(网页总数)。我们用一个以**Score**为类型的一维数组**vector<Score> PageRank**保存所有网页的PageRank值。**

struct Score

{

int WebID;

double score;

};

//初始化PageRank

vector<Score> InitPageRank(vector<int> allWebsite, int num, double score)

{

vector<Score> PageRank(num); //PageRank数组

for (int i = 0; i < num; i++)

{

PageRank[i].WebID = allWebsite[i];

PageRank[i].score = score;

}

return PageRank;

}

**7、Map-Reduce。**

**分三个函数执行。分别是总计算、Map部分、Reduce部分。**

**在Reduce部分采用【随机跳转】的方法解决dead ends和spider trap节点相关问题。**

//总计算 Map\_reduce

vector<Score> Map\_reduce(vector<WebData> SourceMap[][4], vector<Score> PageRank)

{

//执行100次，保证其收敛性

for (int k = 0; k < 100; k++)

{

int rowNum = PageRank.size();

//16个分块进行map操作

vector< vector<double> > MapValue(rowNum);

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

MapValue = CalcuMap(SourceMap[i][j], PageRank, MapValue);

}

}

//reduce操作

PageRank = CalcuReduce(MapValue, PageRank);

}

sort(PageRank.begin(), PageRank.end(), cmpPGRK); //排序

return PageRank;

}

//Map

vector< vector<double> > CalcuMap(vector<WebData> SourceMap, vector<Score> pagerank, vector< vector<double> > MapValue)

{

int nPosition = 0;

//相关数据存入

for (int i = 0; i < SourceMap.size(); i++) //遍历所有源网页

{

//NowScore = pagerank[源网页].score / 源网页的出度数

//遍历PageRank查找当前源网页，取出它的位置，得到score

for (; nPosition < pagerank.size(); nPosition++)

{

if (pagerank[nPosition].WebID == SourceMap[i].WebID)

break;

}

for (int j = 0; j < SourceMap[i].Target.size(); j++)

//每一源网页对应的所有目标网页

{

double NowScore = pagerank[nPosition].score / SourceMap[i].NumOfOut;

//这里号码与位置不是一一对应的关系，所以要先获取SourceMap[i].Target[j]应该处于的位置

int position = BinSearch(allWebsite, allWebsite.size(), SourceMap[i].Target[j]);

MapValue[ position ].push\_back(NowScore);

}

}

return MapValue;

}

//Reduce

vector<Score> CalcuReduce(vector< vector<double> > MapValue, vector<Score> pagerank)

{

int rowNum = pagerank.size();

vector<Score> NewPageRank(rowNum);

//遍历PageRank中的每一个网页，填写新的PageRank值

//遍历MapValue，搜集该目标网页的所有分数

for (int k = 0; k < MapValue.size(); k++)

{

int Name = pagerank[k].WebID;

double sco = 0;

for (int i = 0; i < MapValue[k].size(); i++)

sco += MapValue[k][i];

//考虑dead ends和spider trap的计算方法

double TrueSco = 0.85\*sco + 0.15 / rowNum;

NewPageRank[k].WebID = pagerank[k].WebID;

NewPageRank[k].score = TrueSco;

}

return NewPageRank;

}

**8、按要求将结果写入result.txt**

//写结果

void WriteResult(vector<Score> PageRank)

{

fstream file1;

file1.open("result.txt");

for (int i = 0; i < 100; i++)

{

file1 << PageRank[i].WebID << " " << PageRank[i].score << endl;

}

file1.close();

}

**9、主函数**

int main()

{

load();

cout << "数据wikidata已读取完毕" << endl;

sort(Data.begin(), Data.end(), cmp2);

fourXfour\_Map();

cout << "已分块完毕" << endl;

vector<int> allTarget = AllTarget();

allWebsite = AllWeb(allTarget);

cout << "所有网页已获取完毕" << endl;

int num = allWebsite.size();

double score = 1.0 / num;

vector<Score> PageRank = InitPageRank(allWebsite, num, score);

cout << "PageRank已初始化完毕" << endl;

PageRank = Map\_reduce(Map, PageRank);

cout << "Map\_reduce已执行完毕" << endl;

WriteResult(PageRank);

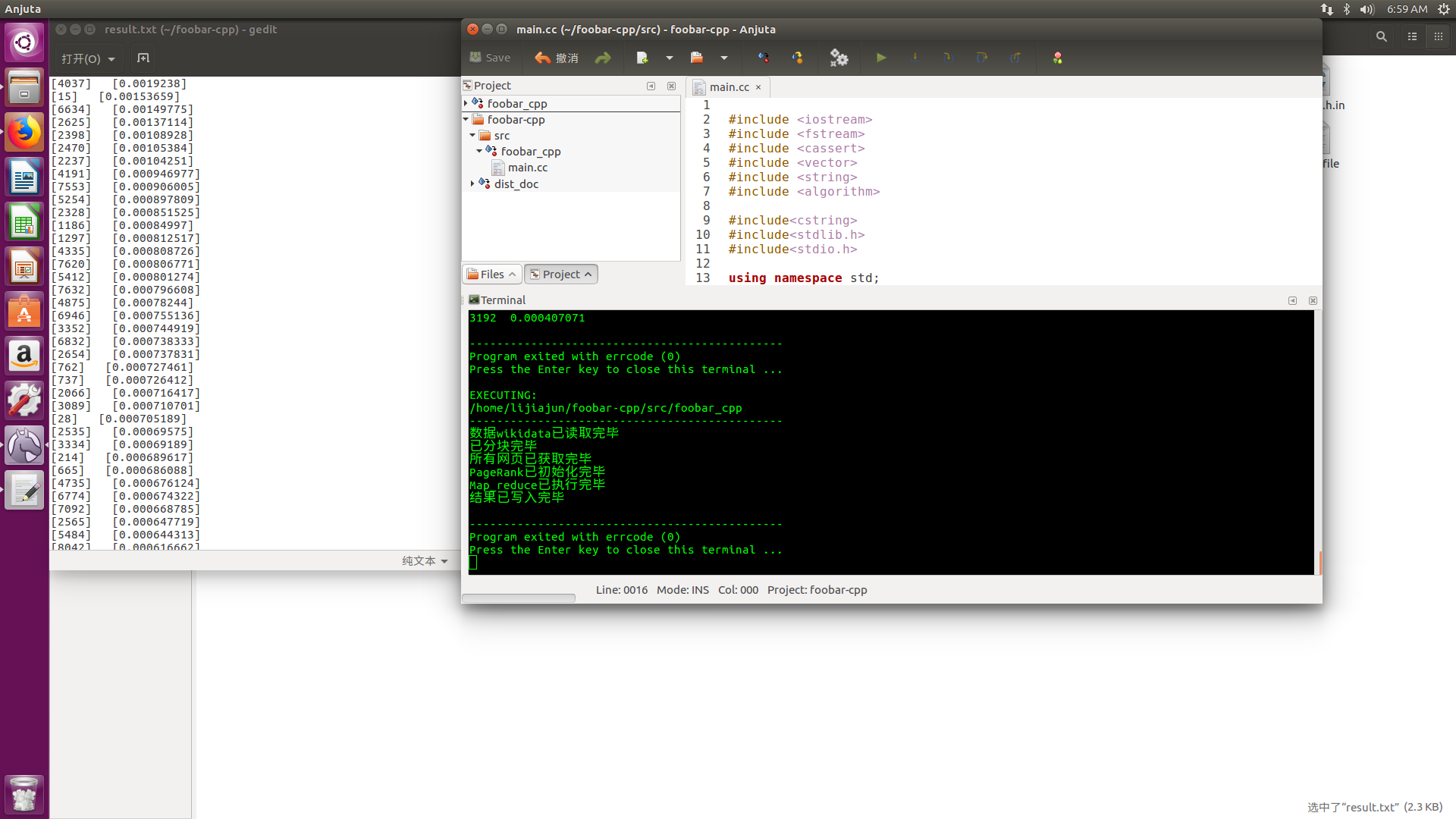
cout << "结果已写入完毕" << endl;

return 0;

}

## 实验结果

* 1. LINUX操作系统/Ubuntu上程序运行截图：



* 1. 前100名的网页及其对应的pagerank值

[4037] [0.0019238]

[15] [0.00153659]

[6634] [0.00149775]

[2625] [0.00137114]

[2398] [0.00108928]

[2470] [0.00105384]

[2237] [0.00104251]

[4191] [0.000946977]

[7553] [0.000906005]

[5254] [0.000897809]

[2328] [0.000851525]

[1186] [0.00084997]

[1297] [0.000812517]

[4335] [0.000808726]

[7620] [0.000806771]

[5412] [0.000801274]

[7632] [0.000796608]

[4875] [0.00078244]

[6946] [0.000755136]

[3352] [0.000744919]

[6832] [0.000738333]

[2654] [0.000737831]

[762] [0.000727461]

[737] [0.000726412]

[2066] [0.000716417]

[3089] [0.000710701]

[28] [0.000705189]

[2535] [0.00069575]

[3334] [0.00069189]

[214] [0.000689617]

[665] [0.000686088]

[4735] [0.000676124]

[6774] [0.000674322]

[7092] [0.000668785]

[2565] [0.000647719]

[5484] [0.000644313]

[8042] [0.000616662]

[4310] [0.00061048]

[5423] [0.000591952]

[1211] [0.000591933]

[3456] [0.000591635]

[2657] [0.000570057]

[5404] [0.000569541]

[5233] [0.000568634]

[4712] [0.000560327]

[271] [0.000553801]

[4828] [0.000543096]

[5079] [0.000542487]

[4261] [0.000536888]

[5210] [0.000534606]

[8163] [0.000527493]

[6914] [0.000526345]

[3459] [0.000525012]

[2285] [0.000518476]

[1549] [0.000518412]

[1842] [0.000515772]

[4666] [0.000506061]

[993] [0.000501183]

[3084] [0.000494182]

[3562] [0.000491973]

[1026] [0.000489755]

[5123] [0.000486415]

[2958] [0.00048393]

[3537] [0.000477825]

[3117] [0.000477614]

[2576] [0.000476042]

[2643] [0.000468178]

[922] [0.00046352]

[5022] [0.000462636]

[299] [0.000461018]

[1855] [0.000459937]

[3897] [0.000458926]

[2651] [0.000458788]

[4110] [0.000457853]

[282] [0.00044881]

[4600] [0.000446989]

[1031] [0.000446468]

[4687] [0.000446156]

[7699] [0.000444851]

[4536] [0.000444502]

[2871] [0.000444299]

[2746] [0.000444223]

[3443] [0.000443748]

[3755] [0.00044319]

[1385] [0.000442371]

[3568] [0.000439157]

[5459] [0.000438286]

[5543] [0.000438072]

[4400] [0.000435598]

[2485] [0.000429906]

[1633] [0.000424721]

[7890] [0.000423497]

[3976] [0.000422978]

[4256] [0.000422407]

[1726] [0.000416045]

[3238] [0.00041367]

[2323] [0.000412033]

[6784] [0.000410091]

[3034] [0.000408795]

[3192] [0.000407071]

## 实验结果分析

1. 最终对PageRank数组中所有网页的PageRank值求和结果为0.415036，可知数据集中dead ends节点较多，PageRank整体损失值较大，可能会造成预测偏差。
2. PageRank值最大的网页4037的PageRank值仅为0.0019238，可知矩阵稀疏，网页间连接不太紧密。