



南开大学
Nankai University

南 开 大 学
网 络 空 间 安 全 学 院
大数据计算与应用实验报告

实验一：计算给定数据集上 pagerank

罗功成 1910487

年级：2019 级

专业：信息安全

2022 年 5 月 3 日

一、 实验目的

对给定数据集，考虑 dead ends 和 spider track 节点，优化稀疏矩阵，实现分块计算，最终编码实现 pagerank 算法，并报告前 100 名 NodeID 及其 PageRank 分数。

可以选择不同的参数，例如 teleport 参数，以比较不同的结果。

二、 实验环境

windows10+vs2022+release 编译。

三、 数据集说明

1. 数据集文件应与.sln 文件或 exe 文件放在同一个文件夹（BigdataPagerank）下。

2. 对于 VS2022C++ 的文件读取，数据集文件的编码形式应为 UTF-8，可用 txt 中另存为的方式将数据集进行修改编码形式。

3. 可执行文件采用 release 方式编译，生成后的 exe 放在 BigdataPagerank->x64->Release

程序源码：BigdataPagerank.cpp,pagerank.h

可执行文件：BigdataPagerank.exe

实验报告：.pdf 文档

四、 程序运行说明

1. 可以将 release 版本的 exe 文件和 data 文件放在同一个文件夹后，直接点击运行 exe.

2. 还可以用 windows 版本的 VS2022 打开 BigdataPagerank.sln 文件，选中开始执行（不调试）的方式运行程序。

经实测，在 win10,win11 平台下的其他电脑均可正常运行该程序。

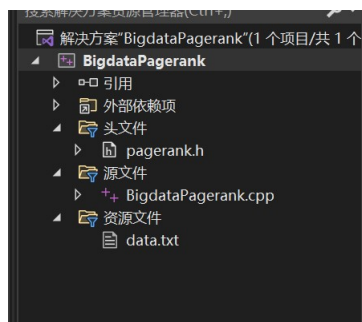


图 1: 实验需要的文件

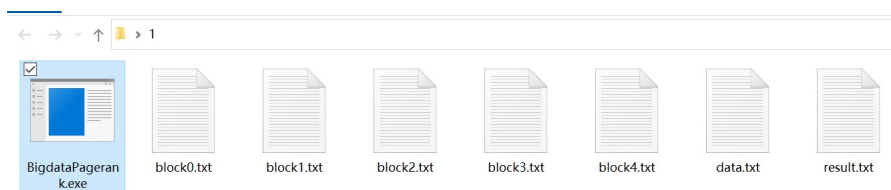


图 2: 可执行文件, 数据集, 分块, 结果

五、 代码实现思路

1. 数据预处理:

- (1) 首先统计节点的最大数量, 从而确定需要切分的块数。
- (2) 设置统计出度, 入度的数组, 输入时在对应节点的出度入度处标记为有效。确定有效节点, 将没有出度入度的孤立节点权重设置为零, 并标记无效。其他节点初始化为 $1/N$
- (3) 有入度无出度的节点, 标记为黑洞, 后面迭代时单独处理。

2. 分块处理:

- (1) 根据计算需要创建对应的块, 在每个块中, 记录形如: [from] [from 的入度] [to], 代表了节点指向关系和 from 节点的初始投票权 (入度), 新加入的数据从链表尾部插入。
- (2) 建立缓冲区 S, 每次从当前对应的块中读入部分数据进行计算。

3. 蜘蛛网陷阱和黑洞的处理:

在本次实验中, 考虑到黑洞和蜘蛛网陷阱的同时存在, 采用的是对 $1-\beta$ 改进后的随机游走方式。

- (1) 首先在数据预处理过程先判断有哪些点是黑洞, 对于黑洞节点, 采用全连接 $1/N$ 的方式来链接;

(2) 对于非黑洞但包含蜘蛛网陷阱的节点，可以做到先按大概率 (β) 按照原定的网络路径走，小概率 ($1-\beta$) 进行随机游走的方式。

4. 迭代过程:

- (1) 对无效节点，更新后的值设为 0;
- (2) 对有效节点，按照 3 中的方式进行反复迭代处理。
- (3) 当迭代前后差值收敛到设定阈值时，停止循环。

5. 排名:

- (1) 构建成绩列表，成员形如 [NodeID] [Score] 的形式
- (2) 用 sort 函数进行排序。
- (3) 由于默认是升序排列，这样输出下标不好考虑，因此补充 compare 函数完成降序规则，最后按下标将前一百位输出到 txt 中即可。

六、 核心代码

统计节点总数

```
1  int a[100000];
2  int count1 = 0;
3  // 读取原数据，统计总节点数
4  void countallnode()
5  {
6      ifstream in("data.txt");
7      int fromNode, toNode;
8      while (!in.eof())
9      {
10         a[0] = 0;
11         for (int j = 0; j < count1; j++)
12         {
13             if (fromNode != a[j])
14             {
15                 count1++;
16                 fromNode = a[count1];
17             }
18             if (toNode != a[j])
19             {
20                 count1++;
```

```

21         toNode = a[count1];
22     }
23 }
24 }
25 }

```

有效节点和黑洞识别

```

1  for (int i = 0; i < count1; i++)
2      {
3          counttoNode[i] = 0;
4          valid[i] = false; //预设各节点均为无入度的
5          valid1[i] = false; //预设各节点均为无出度的
6          blackhole[i] = false; //预设各节点均不是黑洞的
7      }
8  ifstream in("data.txt");
9  int fromNode, toNode;
10 //读取原文件，检测节点的有效性和出度
11 while (!in.eof())
12 {
13     in >> fromNode >> toNode;
14     valid[fromNode - 1] = true;
15     valid1[toNode - 1] = true;
16     //valid[toNode - 1] = true;
17     counttoNode[fromNode - 1]++;
18 }
19
20 int count5 = 0;
21 //统计有效节点数量
22 for (int i = 0; i < count1; i++)
23 {
24     if (valid[i] || valid1[i])
25     {
26
27         count2++;
28     }
29
30     if (valid[i] && (valid1[i] == false))
31     {
32         count5++; //统计黑洞数量
33         blackhole[i] = true;
34     }
35 }
36
37 for (int i = 0; i < count1; i++)
38 {
39     if (valid[i] || valid1[i])
40     {
41         Wtn[i] = 1 / count2; //初始时有效节点平分归一

```

化的权重

```

42         }
43         else
44         {
45             Wtn[i] = 0; //无效节点权重为0
46         }
47         Wtn1[i] = 0; //等待更新权重的初始化
48     }

```

分块处理

```

1  void blockSaving()
2  {
3      count3 = count1 / blocksize + 1;
4      ofstream* out = new ofstream[count3];
5      char buf[100];
6      for (int i = 0; i < count3; i++)
7      {
8          sprintf_s(buf, "block%d.txt", i);
9          out[i].open(buf);
10     }
11     ifstream in("data.txt");
12     int fromNode, toNode;
13     int current = -1; //头节点，放在第一个节点之前
14     int count6 = 0; //记录当前记录了多少条，判断是否要换到下一个块记录数据
15     list<int> LinkList;
16     while (!in.eof())
17     {
18         in >> fromNode >> toNode;
19         if (current != fromNode)
20         {
21             int t;
22             while (!LinkList.empty())
23             {
24                 t = LinkList.front();
25                 out[current / blocksize] << current << " " << counttoNode[current -
26                     1] << " " << t << endl;
27                 LinkList.pop_front();
28             }
29             current = fromNode;
30             count6 = 0;
31         }
32         count6++;
33         LinkList.push_back(toNode); //数据写入链表尾部
34     }
35     cout << "完成分块" << endl;
36     cout << endl;
37

```

38 }

随机游走解决蜘蛛网陷阱和黑洞

```

1  Wtn1[toNode - 1] = Wtn1[toNode - 1] + Wtn[fromNode - 1] / votes * beta;
2  if (m!=toNode)//当数据集中来到新的toNode时, 进行一次随机游走
3  {
4      //1.直接以1-beta的概率, 在全体数据集上进行随机游走, 解决蜘蛛网陷阱
5      //Wtn1[toNode - 1] = Wtn1[toNode - 1] + (1 - beta) / count1;
6      //2.改为以1-sblackhole概率, 在解决蜘蛛网陷阱的基础上解决黑洞,
7      //sblackhole=rold/di
8      //int s= Wtn1[toNode - 1] = Wtn1[toNode - 1] + Wtn[fromNode - 1] /
9      //votes * beta;
10     // //r=r+(1-r)/N, r=0时代表黑洞
11     /*Wtn1[toNode - 1] = Wtn1[toNode - 1] + (1 - Wtn1[toNode - 1]) /
12     count1;*/
13     //3.对非黑洞部分1-beta, 对黑洞部分1/N
14     if (blackhole[toNode-1]==false)
15     Wtn1[toNode - 1] = Wtn1[toNode - 1] + (1 - beta) / count1;
16     else
17     {
18         //Wtn1[toNode - 1] = Wtn1[toNode - 1] + (1 - Wtn1[toNode -
19         1]) / count1;
20         Wtn1[toNode - 1] = 1 / count1;
21     }
22     m = toNode; //更新临时值m

```

迭代运算

```

1  bool flag = true; //迭代标志
2  int count4 = 0; //迭代次数
3  while (flag) //没有结束前反复迭代接近阈值
4  {
5      count4++;
6
7      for (int i = 0; i < count3; i++)
8      {
9          char s[100];
10         sprintf_s(s, "block%d.txt", i);
11         ifstream in(s);
12         int fromNode, votes, toNode; //从某节点投了n票给另一个
13         节点
14         while (!in.eof())
15         { //随机游走解决蜘蛛网陷阱和黑洞部分的迭代更新}
16
17             flag = Isend();
18
19         bool Isend()

```

```

19 {
20     double New = 0.0, Old = 0.0;
21     bool flag = false; //标志位, 标记是否可以结束迭代
22     for (int i = 0; i < count1; i++)
23     {
24         Old = Old + Wtn1[i];
25     }
26     for (int i = 0; i < count1; i++)
27     {
28         if (valid[i] == false && valid1[i] == false)
29         {
30             Wtn1[i] = 0; //无效节点更新后的权重还设为0
31             continue;
32         }
33         else
34         {
35             Wtn1[i] += (1 - Old) / count2;
36             if (Wtn[i] - Wtn1[i] > sigma || Wtn1[i] - Wtn[i] >
37                 sigma) //迭代前后和阈值sigma对比
38             {
39                 flag = true; //差值较大, 设置标志位, 再次循环
40             }
41             Wtn[i] = Wtn1[i]; //将上一轮迭代数据更新
42             Wtn1[i] = 0;
43             New = New + Wtn1[i];
44         }
45     }
46     return flag;
47 }

```

分数排名

```

1 struct ScoreList
2 {
3     int NodeId;
4     double Score;
5 };
6 bool Compare(ScoreList r1, ScoreList r2) //添加比较函数实现sort降序
7 {
8     return r1.Score > r2.Score;
9 }
10
11 ScoreList* L; //构建分数排名列表
12 void topRankCount()
13 {
14     L = new ScoreList[count1];
15     for (int i = 0; i < count1; i++)
16     {

```



```
17         L[i].NodeId = i;  
18         L[i].Score = Wtn[i];  
19     }  
20     sort(L, L + count1, Compare); //降序  
21     //sort(p, p + count1); 升序排列  
22  
23 }
```

七、 实验结果

```
C:\Users\admin\Desktop\pagerank\x64\Release\BigdataPagerank.exe  
节点总数为: 8297  
数据预处理  
将孤立点权重设为0, 统计有效节点  
有出度/入度节点有6263个  
其中有黑洞4226个  
统计有效节点完成  
每块大小为2000数据共可分为5块  
开始分块  
完成分块  
开始迭代  
经过了31次迭代后收敛至设定阈值1e-06  
节点排列完成  
将Top 100 NodeID以[NodeID] [Score]的形式保存到result.txt  
请按任意键继续. . .
```

图 3: 实验命令行输出效果

```
4037 0.004729  
6634 0.00447743  
2625 0.00404184  
15 0.00305366  
2398 0.00300254  
6946 0.00289917  
3089 0.00268374  
2328 0.00252276  
5412 0.0024713  
7632 0.00246009  
4191 0.00241843  
3352 0.00240965  
7553 0.00240803  
6832 0.00228986  
3456 0.00221823
```

图 4: 输出结果的 txt 中部分截图

八、 结果分析

1. 节点中有很多孤立节点, 并且还有很多黑洞。通过预处理数据的方式可以减少运算量, 提高效率。

2. 分块不同时，计算结果略有不同，排名几个 PageRank 接近的节点之间位置可能有所变化
3. 设定不同的阈值时，PageRank 值和迭代次数有所不同，排名靠前的节点相对位置不变，后面几个 PageRank 接近的某几个节点位置可能有所变化。

下面进行对比试验进行具体说明：

九、 对比实验

实验中，块大小设置为 2000， $\beta=0.85$, $\sigma=0.000001$

1. 切分为不同的块大小：

大小为 1000 时的输出：

```
节点总数为: 8297
数据预处理
将孤立点权重设为0, 统计有效节点
有出度/入度节点有6263个
其中有黑洞4226个

统计有效节点完成

每块大小为1000数据共可分为9块
开始分块
完成分块

开始迭代
经过了127次迭代后收敛至设定阈值1e-13

节点升序排列完成

将Top 100 NodeID以[NodeID] [Score]的形式保存到result.txt
```

图 5: 输出命令行截图

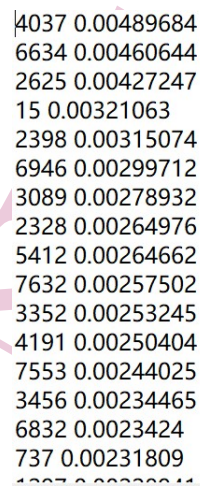
```
6634 0.00499252
4037 0.00464055
2625 0.00402673
6946 0.0032733
2398 0.00317044
15 0.00305125
3089 0.0028924
7632 0.00265767
7553 0.00262465
5412 0.00260427
4191 0.00254085
2328 0.00249352
3352 0.00248125
6832 0.00246533
4712 0.00242554
3456 0.00234626
```

图 6: 输出结果的 txt 中部分截图

2. 更换不同的 β , σ 值: $\beta=0.9, \sigma=0.0001$ 

```
Microsoft Visual Studio 调试控制台
节点总数为: 8297
数据预处理
将孤立点权重设为0, 统计有效节点
有出度/入度节点有6263个
其中有黑洞4226个
统计有效节点完成
每块大小为2000数据共可分为5块
开始分块
完成分块
开始迭代
经过了8次迭代后收敛至设定阈值0.0001
节点升序排列完成
将Top 100 NodeID以[NodeID] [Score]的形式保存到result.txt
```

图 7: 输出命令行截图



```
4037 0.00489684
6634 0.00460644
2625 0.00427247
15 0.00321063
2398 0.00315074
6946 0.00299712
3089 0.00278932
2328 0.00264976
5412 0.00264662
7632 0.00257502
3352 0.00253245
4191 0.00250404
7553 0.00244025
3456 0.00234465
6832 0.0023424
737 0.00231809
4227 0.00222214
```

图 8: 输出结果的 txt 中部分截图

可见, 比实验数据普遍偏大了, 且离散程度变大了一些。

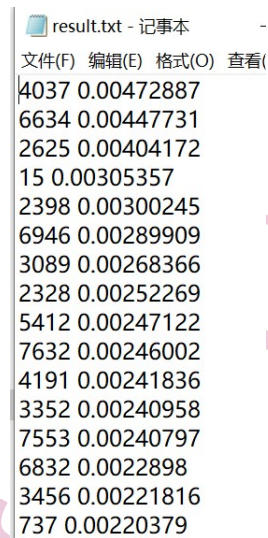
可能的因素有:

- (1) σ 设置较大, 导致收敛效果还不够好;
- (2) 同时 β 值较大, 随机游走部分偏少了。

 $\beta=0.85, \sigma=0.0000000001$

```
节点总数为: 8297
数据预处理
将孤立点权重设为0, 统计有效节点
有出度/入度节点有6263个
其中有黑洞4226个
统计有效节点完成
每块大小为2000数据共可分为5块
开始分块
完成分块
开始迭代
经过了130次迭代后收敛至设定阈值1e-13
节点升序排列完成
将Top 100 NodeID以[NodeID] [Score]的形式保存到result.txt
C:\Users\admin\Desktop\BigdataPagerank\x64\Release\BigdataPagerank.exe (进程 56280) 已退出, 代码为 0。
```

图 9: 实验命令行输出效果



```
result.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V)
4037 0.00472887
6634 0.00447731
2625 0.00404172
15 0.00305357
2398 0.00300245
6946 0.00289909
3089 0.00268366
2328 0.00252269
5412 0.00247122
7632 0.00246002
4191 0.00241836
3352 0.00240958
7553 0.00240797
6832 0.0022898
3456 0.00221816
737 0.00220379
```

图 10: 输出结果的 txt 中部分截图

虽然收敛效果变好了一点点, 但是提升效果很有限, 且效率下降明显。

3.1-s vs 1-beta:

1-beta: 没有考虑黑洞存在。

1-S 的思想是, 既然黑洞权重为零, 那么 S 是 PageRank 值 (Rold) 时, 黑洞 S , rold 就为 0, $R_{new} = R_{old} + (1-S)/N$ 时就可以做到更新为 $1/N$ 。同时 1-S 和 1-beta 数值差不多时, 对蜘蛛网陷阱来说, 也可以做到大概率按照原来路线走, 小概率随机游走。这样完成了对两个问题同时解决。

为什么不直接用 1-S 的方式直接完成对蜘蛛网陷阱和黑洞的同时处理?

因为本实验中算得的 PageRank 值较小, 最高只有 0.004. 如果直接按照 1-S (S 是 PageRank 值) 时, $(1-S)$ 的数值达到了 0.996-0.999, 相当于比 β 还高的概率去走随机游走; 在解决蜘蛛网陷阱中, β 设为 0.8-0.9 时, 1-beta 为 0.1-0.2, 明显低于 0.99. 这显然是不符合对网络中应当以大概率按原来路径, 拿出一小部分来随机散布的思想。

因此，需先判断是否为黑洞。是，设为 $1/N$ ；不是，按 β 和 $1-\beta$ 处理。

不同策略的结果输出文件见作业文件夹中。

十、 总结和收获

1. 在编程实践中，充分训练并掌握了 pagerank 的相关知识和算法实现，对于黑洞，蜘蛛网陷阱等网络中存在的问题有了更为深入的认知。
2. 对于分块计算、循环迭代提升精度的算法思想得到了学习和提高。
3. 通过一系列的对比实验，对随机游走的实现方式，参数的选择等方面有了新的理解。

MINIB