PageRank实验报告

一、数据集说明

本次实验中使用的数据集为示例中给出的数据集 `Data.txt`，包含 `node` 8297个， `link` 135737条。其中出度为0的节点共计2187个，详细信息存储于 `degree.txt`

Data.txt - 记事本

文件(F) 编辑(E) 格式(

```
2222 5878
1240 1249
5880 3433
6479 6339
```

degree.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
1371    out_degree: 31    in_degree10
1372    out_degree: 28    in_degree19
1373    out_degree: 18    in_degree21
1374    out_degree: 28    in_degree18
```

二、实验原理

1. 数量假设

当在网页模型图中，一个网页接受到的其他网页指向的入链（in-links）越多，说明该网页越重要。

2. 质量假设

当一个质量高的网页指向（out-links）一个网页，说明这个被指的网页重要。

3. 基本原理

一般 PageRank 的定义意味着互联网浏览者，按照以下方法在网上随机游走：任意一个网页上，浏览者或者以概率 `teleport` 决定按照超链接随机跳转，这时以等概率从连接出去的超链接跳转到下一个网页；或者以概率 `(1-teleport)` 决定完全随机跳转，这时以等概率 `1/n` 跳转到任意一个网页。第二个机制保证从没有连接出去的超链接的网页也可以跳转出。

三、关键代码解析

`Init` 函数

```
1 int Init(const char* filename)
2 {
3         fstream input(filename);
4         Node_ID from = 0;
5         Node_ID to = 0;
6         while (!input.eof() && input >> from >> to)
```

```
        {
                in_page[to].push_back(from);
                out_page[from].push_back(to);
                Node_num = max(Node_num, max(from, to));
        }
        input.close();
        in_degree.resize(Node_num + 1);
        out_degree.resize(Node_num + 1);
        for (unsigned int i = 0; i <= Node_num; i++)
        {
                in_degree[i] = in_page[i].size();
                out_degree[i] = out_page[i].size();
        }
        unsigned int count = Node_num;
        for (unsigned int i = 0; i <= Node_num; i++)
        {
                if (in_degree[i] == 0 && out_degree[i] == 0)
                {
                        count--;
                }
        }
        return count + 1;
}
```

功能：该函数负责从指定的文件名（filename）读取网络链接数据，构建网络图结构，并统计节点的入度、出度等信息。
具体实现：
打开文件：使用std::fstream打开指定的文本文件（Data.txt）。
读取数据：逐行读取文件内容，每行包含两个整数（from和to），代表一条从节点from到节点to的有向边。将这些边信息添加到in_page和out_page映射表中，分别表示节点的入链和出链。同时，更新全局变量Node_num以存储当前网络中的最大节点ID。
计算节点入度和出度：关闭文件后，根据已构建的网络图，为所有节点（包括未出现在边数据中的节点）分配入度和出度数组，并按节点ID填充其实际入度和出度值。
统计有效节点数量：遍历节点的入度和出度数组，减去那些入度和出度均为0的孤立节点数量，最终返回有效节点数量（即参与PageRank计算的节点数量）。
输出：返回一个整数，表示有效节点的数量。

`get_rank` 函数

```
void get_rank(int n, double teleport)
{
        old_rank.resize(Node_num + 1, 0);
```

```
        new_rank.resize(Node_num + 1, 1.0 / n);
        double loss = 0.0;
        do
        {
                loss = 0.0;
                for (auto i : in_page)
                {
                        new_rank[i.first] = 0.0;
                        for (int j = 0; j < i.second.size(); j++)
                        {
                                if (out_degree[i.second[j]] != 0)
                                {
                                        new_rank[i.first] += teleport * (old_rank[i.second[j]] / out_degree[i.second[j]]);
                                }
                                else
                                {
                                        new_rank[i.first] += 0;
                                }
                        }
                }
                double S = 0.0;
                for (auto i : new_rank)
                {
                        S += i;
                }
                for (int i = 0; i < new_rank.size(); i++)
                {
                        new_rank[i] += (1.0 - S) / n;
                }
                for (int i = 0; i < new_rank.size(); i++)
                {
                        double temp = new_rank[i] - old_rank[i];
                        if (temp > 0)
                        {
                                loss += temp;
                        }
                        else
                        {
                                loss -= temp;
                        }
                }
                for (int i = 0; i < new_rank.size(); i++)
                {
                        old_rank[i] = new_rank[i];
                }
                cout << "current loss: " << loss << endl;
```

```
        } while (loss > STOPLOSS);
        cout << "final loss: " << loss << endl;
        ofstream result("result.txt");
        for (auto i : in_page)
        {
                result << i.first << "\t" << new_rank[i.first] << endl;
        }
        result.close();
        return;
}
```

功能：该函数实现Google PageRank算法，计算网络中节点的排名。参数n表示有效节点数量，teleport是随机跳转概率。
具体实现：
初始化：为所有节点分配初始排名值。所有节点的初始PageRank值设为1/n，存储在new_rank数组中；old_rank数组用于保存上一轮迭代的PageRank值。
迭代计算：执行以下循环，直到PageRank值变化小于给定阈值（STOPLOSS）为止：
a. **更新PageRank值**：对于每个节点，根据其入链节点的PageRank值及其出链数量重新计算其PageRank值。同时，根据随机跳转概率teleport进行调整。对于没有出链的节点，其贡献为0。
b. **归一化**：计算当前PageRank值总和，将其与n相减后平均分配给所有节点，确保PageRank值总和为1。
c. **计算损失**：计算当前PageRank值与上一轮PageRank值之间的差异（绝对值之和），作为损失值。
d. **更新旧值**：将当前PageRank值复制到old_rank数组，准备下一轮迭代。
输出结果：将最终PageRank值写入到result.txt文件中，格式为“节点ID”和“PageRank值”的对应关系。
输出：将PageRank结果写入到文件result.txt中。

`gettop100` 函数

```
void gettop100()
{
        for (int i = 0; i < new_rank.size(); i++)
        {
                Node_score[i] = new_rank[i];
        }
        vector<PAIR> nodes_top(Node_score.begin(), Node_score.end());
        sort(nodes_top.begin(), nodes_top.end(), cmp_value);
        ofstream topnode("top100.txt");
        for (int i = 0; i < 100; i++)
        {
                topnode << nodes_top[i].first << '\t' << nodes_top[i].second
<< endl;
```

```
        }
        topnode.close();
        return;
}
```

功能：该函数从已计算得到的PageRank值中，提取排名前100的节点及其得分，并将结果写入到top100.txt文件中。
具体实现：
创建排序键值对：将new_rank数组中的节点ID及其对应的PageRank值封装成PAIR类型（std::pair<Node_ID, double>），并存入Node_score映射表中。
排序：使用std::sort对节点PageRank值进行降序排列。
输出：将排名前100的节点ID及其PageRank值写入到top100.txt文件中，每行包含一个节点的ID和PageRank值。
输出：将排名前100的节点及其得分写入到文件top100.txt中。

四、实验结果及分析

程序运行后得到对应 `teleport` 下的结果，分别为 `result.txt、degree.txt、top100.txt` 。

其中 `result.txt` 为所有节点及其对应的 `PageRank` 值， `degree.txt` 为所有节点对应的 `入度` 与 `出度` ， `top100.txt` 为 `PageRank` 值从高到低的前一百个节点及其 `PageRank` 值。

此处我们以两组数据进行对比说明（teleport分别为0.85、0.90）

`degree.txt` ：

该文件标明了各节点的对应的 `入度` 与 `出度` ，且与 `teleport` 取值无关，只随初始数据集而变化

`result.txt` ：

result(0.85).txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助

2726	8.07596e-05
2727	7.54442e-05
2728	9.54346e-05
2729	8.16516e-05
2730	0.00087186
2731	7.95059e-05
2732	0.000110758
2733	9.26844e-05

result(0.90).txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

2726	7.00643e-05
2727	6.48754e-05
2728	8.32792e-05
2729	7.03671e-05
2730	0.00115655
2731	6.85445e-05
2732	9.75866e-05
2733	8.07906e-05

对比两组数据，可以发现随着 `teleport` 的增加，原数据中 `PageRank` 值较大的增大，较小的减小，即数据更加分散。这个结果是符合预期的， `teleport` 值越大，则 `PageRank` 值更接近原来的网络关系分布，归一化分配的平均值更小，数据更加分散。一个极端的例子是当 `teleport` 为0时，所有节点的 `PageRank` 值都相等。

`top100.txt`：

top100(0.85).txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮

2730	0.00087186
7102	0.000854534
1010	0.000849616
368	0.000835903
1907	0.000830595
7453	0.000820647
4583	0.000817883

top100(0.90).txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

2730	0.00115655
7102	0.00113552
1010	0.00112895
368	0.00111009
1907	0.0011036
7453	0.00108427
4583	0.00108404

对比两组数据，发现随着 `teleport` 的变化，节点 `PageRank` 值的变化符合上一部分的分析，且可以看到改变 `teleport` 的值只会改变各节点的 `PageRank` 值，而不会影响各节点的最终排名。