Report for hw3

**Dataset:**

graph\_1.txt -> graph\_8.txt

graph7 and 8 come from hw1.

**Assignment1 Hits**

**Usage:**

g++ hits.cc -o hits

./hits graph\_1.txt

./hits graph\_2.txt

...

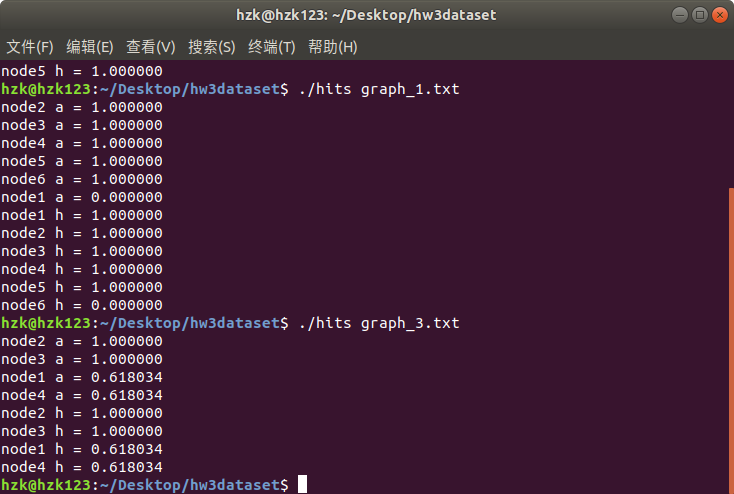
./hits graph\_8.txt

如果想要结果输出到文件 只需要用>重定向

./hits graph\_8.txt > hit\_result8.txt

**Result:**

this result is just a sample , other result need to run code



**Algorithm**

**鏈式前向星**

這份code使用了鏈式前向星來製作鏈表

1.1 结构

这里用两个东西：

1 结构体数组edge存边，edge[i]表示第i条边。

2 head[i]存以i为起点的第一条边(在edge中的下标)

struct EDGE{

int next; //下一条边的存储下标(默认0)

int to; //这条边的终点

int w; //权值

};

EDGE edge[500010];

1.2.增边

若以点i为起点的边新增了一条，在edge中的下标为j.

那么edge[j].next=head[i];然后head[i]=j.

即每次新加的边作为第一条边，最后倒序遍历

 void Add(int u, int v, int w) { //起点u, 终点v, 权值w

//cnt为边的计数，从1开始计

edge[++cnt].next = head[u];

edge[cnt].w = w;

edge[cnt].to = v;

head[u] = cnt; //第一条边为当前边

}

3. 遍历

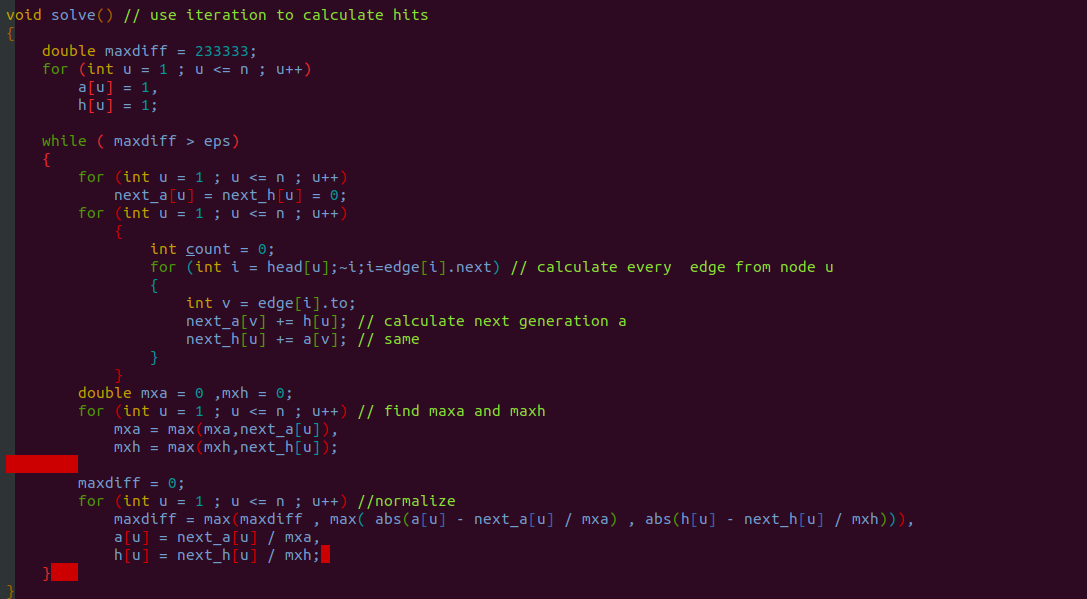
遍历以st为起点的边

for(int i=head[st]; i!=0; i=edge[i].next)

i开始为第一条边，每次指向下一条(以0为结束标志)  （若下标从0开始，next应初始化-1）

**hits计算方法**

对每次迭代，遍历整个图，对图中每个点的出边计算a和h



**Discussion**

通过计算发现最多5次迭代就能达到1e-5的误差。

**Assignment2 Pagerank**

**Usage:**

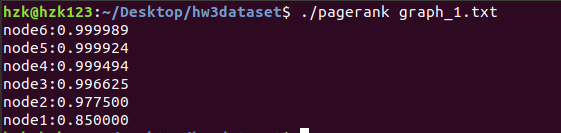
g++ pagerank.cc -o pagerank

./pagerank graph\_1.txt

其他用法同上

**Result**

this result is just a sample , other result need to run code

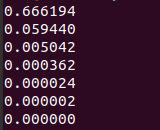


**Algorithm**

1. 使用了链式前向星来制作链表
2. 使用迭代的方式来计算pagerank，在建图时先记录每个点的出度。每次迭代，遍历一次所有节点，对一个节点计算他所有出点的上一轮pagerank迭代的和。遍历完所有节点后还需要normalize一次。

**Discussion**

对于graph5和graph6这两个比较大的图，迭代7次的误差已经小于1e-6。 迭代的收敛速度非常之快，下图为7次迭代时每次的最大变化。



**Assignment3 simrank**

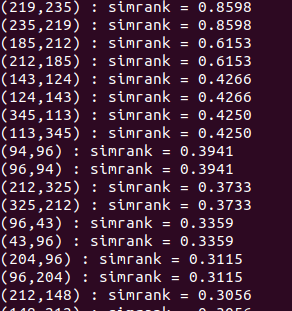
**Usage:**

g++ simrank.cc -o simrank

./simrank graph\_1.txt

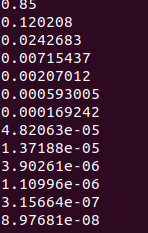
其他用法同上

**Result:**

****

**Discussion**

**观察迭代结果可以发现，对于较大的图8次迭代误差也能收敛在1e-5的范围内，下图为每次迭代后的与上一次的最大差值。**

****

**Compartion and Conclusion in all:**

使用链表来存图，可以发现算法复杂度大概在O((n+E)logK)级别,K为误差，E为图上的边数，这显然比矩阵运算要优秀，矩阵运算每次复杂度为o(n^3),对于n大于100000的情况甚至没有办法存下邻接矩阵。