实验 4　图论算法

1. 实验要求

实验 1：实现求有向图的强连通分量的算法。有向图的顶点数 N 的取值分别为：8、16、32、64、128、256，弧的数目为 NlogN，随机生成 NlogN 条弧，统计算法所需运行时间，画出时间曲线。

实验 2：实现求所有点对最短路径的 Johnson 算法。生成连通的无向图，图的顶点数 N 的取值分别为：8、16、32、64、128、256，边的数目为 NlogN，随机生成 NlogN 条边，统计算法所需运行时间，画出时间曲线。

2. 实验环境

编译环境：Code::Blocks 16.01
机器内存：8G
时钟主频：2.69GHz

3. 实验过程

Project1：

1. 对每一个 size，随机生成相应 input.txt 文件。
2. 对每一个 size，打开对应的 output1.txt 和 time1.txt 文件。方便起见，此处 output1.txt 文件的文件指针为全局变量。
3. 读取对应输入文件 input.txt，并将输入图存储在邻接矩阵中。
4. 开始计时。
5. 对图进行 DFS，求出各点的到达时间 d 和结束时间 f。
6. 根据各点结束时间 f，对顶点进行排序，按照 f 从大到小的顺序把顶点序号存放在数组 order[v]中。
7. 根据 order[v]的顺序，对 G^T 进行 DFS 遍历。此处不再单独求图 G^T，而是在 DFS 中单独设置一个参数，指定这里要对 G^T 进行遍历，然后 DFS 对边进行检测时，将两个顶点的次序颠倒，并将遍历结果输出到文件 output1.txt 中。
8. 结束计时，计算总时间，存放在文件 time1.txt 中。
9. 关闭文件。

Project2：

1. 对于每一个 size，随机生成输入图，并将图存入邻接矩阵中，通过对该邻接矩阵运行 Bellman-Ford 函数检测图中是否有权值为负的回路，若有，则重新生成一个图，重复以上检测过程。方便起见，这里的图为有向图，边的权值可以为负，且遵循以下要求：
   - 1/n 的边的权值为负的，且绝对值不大于 log n。
   - 剩余的边权值为正，且不大于 n。
2. 打开对应的 output2.txt 和 time2.txt 文件。方便起见，此处 output2.txt 文件的文件指针为全局变量。
3. 开始计时。

4. 在邻接矩阵中加入一个新结点。原有结点标号为 0~v-1，新结点标号为 v。新结点到原有 v 个结点之间路径长均为 0，原有节点不可达新结点。
5. 以新加入的结点 v 为源结点，对新的图进行 Bellman-Ford 分析，得到 v 到每个点的最短路径。
6. 将得到的最短路径存储到 h[0..v]数组中，对结点 0~v-1，修改每条边的权值，新权值 G'[i][j]=G[i][j]+h[i]-h[j]，这样以来，新的权值全部为非负。
7. 根据新的权值，对图的每一个结点 i，都以它为源结点，对图进行 Dijkstra 算法。将得到的 i 到各点的最小路径长度存放在 D[0..v-1][0..v-1]矩阵的第 i 行。即 D[i][j]=d[j]+h[j]-h[i]。并将这条最短路径上 j 的前驱结点序号存在矩阵 P[0..v-1][0..v-1]中，P[i][j]位置，P 矩阵用于之后输出最短路径所经过的结点。
8. 停止计时，计算总时间，存储到文件 time2.txt 中。
9. 根据 D 和 P 矩阵，将每两个结点之间的最短路径长和经过的结点输出到文件 output2.txt 中。
10. 关闭文件。

4. 实验关键代码截图（结合文字说明）

Project1

图用一个全局的邻接矩阵存储，对于结点的每个属性，单独开一个全局数组存储。

DFS：

```
void DFS(int v,int order[v],int graph)
{
    int i;
    for(i=0;i<v;i++)
    {
        color[i]=WHITE;
        parent[i]=NIL;
    }
    Time=0;
    for(i=0;i<v;i++)
    {
        int j=order[i];
        if(color[j]==WHITE)
        {
            if(graph==DFS_GT)
                fprintf(result,"(");
            DFS_visit(v,j,graph);
            if(graph==DFS_GT)
                fprintf(result,")\n");
        }
    }
}
```

DFS-visit：

```
void DFS_visit(int v,int u,int graph)
{
    Time++;
    d[u]=Time;
    color[u]=GRAY;
    int i;
    if(graph==DFS_GT)
        fprintf(result,"%d",u);
    for(i=0;i<v;i++)
    {
        if(color[i]!=WHITE) continue;
        if(graph==DFS_G&&G[u][i]==1)
        {
            parent[i]=u;
            DFS_visit(v,i,graph);
        }
        else if(graph==DFS_GT&&G[i][u]==1)
        {
            parent[i]=u;
            fprintf(result,", ");
            DFS_visit(v,i,graph);
        }

    }
    color[u]=BLACK;
    Time++;
    f[u]=Time;
}
```

SCC：

```
void SCC(int n)
{
     int v=v_size[n];
     int order[v];
     int i;
     for(i=0;i<v;i++) order[i]=i;
     DFS(v,order,DFS_G);
     for(i=0;i<v;i++)
     {
          int j=f[i]/2;
          order[v-j]=i;
     }
     DFS(v,order,DFS_GT);
}
```

Project2

输出最短路径：（使用递归函数）

```
void print_path(int i,int j)
{
    if(i==j)
    {
        fprintf(f,"%d",i);
        return;
    }
    int p=P[i][j];
    if(p==i)
    {
        fprintf(f,"%d, %d",i,j);
        return;
    }
    print_path(i,p);
    fprintf(f,", %d",j);
}
```

Dijkstra：

```
void dijkstra(int v,int s)
{
    init_single_source(v,s);
    int q[v];
    int i;
    for(i=0;i<v;i++) q[i]=i;
    for(i=v;i>0;i--)
    {
        int j;
        int min=d[i-1];
        int min_ind=i-1;
        for(j=0;j<i-1;j++)
            if(min>d[j])
            {
                min=d[q[j]];
                min_ind=j;
            }
        int temp=q[min_ind];
        q[min_ind]=q[i-1];
        q[i-1]=temp;
        for(j=0;j<v;j++)
        {
            if(G[temp][j]<MAX)
            d_relax(s,temp,j);
        }
    }
}
```

Johnson：

```
void Johnson(int n)
{
    int v=v_size[n];
    int s=v,i,j;
    for(i=0;i<v;i++)
    {
        G[s][i]=0;
        G[i][s]=INF;
    }
    G[s][s]=INF;

    if(Bellman_Ford(v+1,s)==FALSE)
    {
        printf("graph contains a negative weight cycle\n");
        exit(1);
    }
    for(i=0;i<=v;i++)
    {
        h[i]=d[i];
    }
    for(i=0;i<=v;i++)
    for(j=0;j<=v;j++)
    {
        if(G[i][j]<MAX)
        {
            G[i][j]=G[i][j]+h[i]-h[j];
        }
    }
    for(i=0;i<v;i++)
    {
        dijkstra(v,i);
        for(j=0;j<v;j++)
            D[i][j]=d[j]+h[j]-h[i];
    }
}
```

Bellman-Ford：

```
int Bellman_Ford(int v,int s)
{
    init_single_source(v,s);
    int i,j;
    for(i=0;i<v;i++)
    {
        int k;
        for(j=0;j<v;j++)
        for(k=0;k<v;k++)
        {
            if(G[j][k]<MAX)
                relax(j,k);
        }
    }
    for(i=0;i<v;i++)
        for(j=0;j<v;j++)
        if(G[i][j]<MAX)
        {
            if(d[j]>d[i]+G[i][j])
                return FALSE;
        }

    return TRUE;
}
```

5. 实验结果、分析（结合相关数据图表分析）

	v	e	time1/ms				time2/ms			
			1	2	3	average	1	2	3	average
size1	8	24	0.0031	0.0032	0.0031	0.00313	0.006	0.012	0.006	0.0080
size2	16	64	0.0078	0.0078	0.0078	0.00780	0.058	0.060	0.054	0.0573
size3	32	160	0.0187	0.0156	0.0187	0.01767	0.356	0.374	0.380	0.3700
size4	64	384	0.0547	0.0609	0.0594	0.05833	2.540	2.570	2.668	2.5927
size5	128	896	0.1687	0.1703	0.1734	0.17080	16.606	16.936	16.910	16.8173
size6	256	2048	0.5640	0.5485	0.5531	0.55520	117.284	116.458	116.706	116.8160

上图为 project1 和 project2 的运行时间。下面首先对获取数据的方式进行说明：

1. project1 中的时间包括求强连通分量和将它们存储到对应输出文件中的总时间。此外，为

了数据的准确性，这里把该过程循环 10000 次，再将总时间除以 10000，来得到每次求强连通分量的时间。这 10000 次所用的输入数据不变，故重新生成输入数据，再重复以上过程两次，求平均值。

2. project2 中的时间仅为运行 Johnson 算法的时间。同样，这里将该过程循环 10000 次，再将总时间除以 10000，以提高结果的精确度。这 10000 次所用的输入数据不变，故重新生成输入数据，再重复以上过程两次，求平均值。

数据分析：

1. 求图的强连通分量理论时间复杂度为 O(V+E)

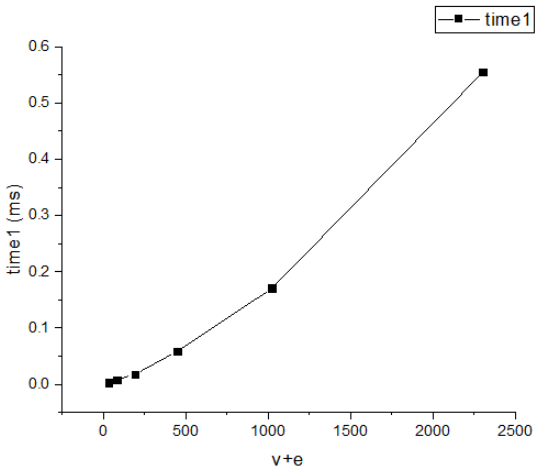
以 V+E 的值为横坐标，time1 为纵坐标作图如下：

对它进行 linear fitting 的结果为：

	Intercept		Slope		Statistics
	Value	Standard Error	Value	Standard Error	Adj. R-Square
time1	-0.03001	0.01768	2.43374E-4	1.68565E-5	0.97647

即 time1 = 2.43374E-4 (V+E) – 0.03001，r-square 的值非常接近 1，即线性拟合结果很好，time1 和 V+E 成线性关系。符合 time1=O(V+E)的理论结果。

v+e	time1
32	0.00313
80	0.00780
192	0.01767
448	0.05833
1024	0.17080
2304	0.55520



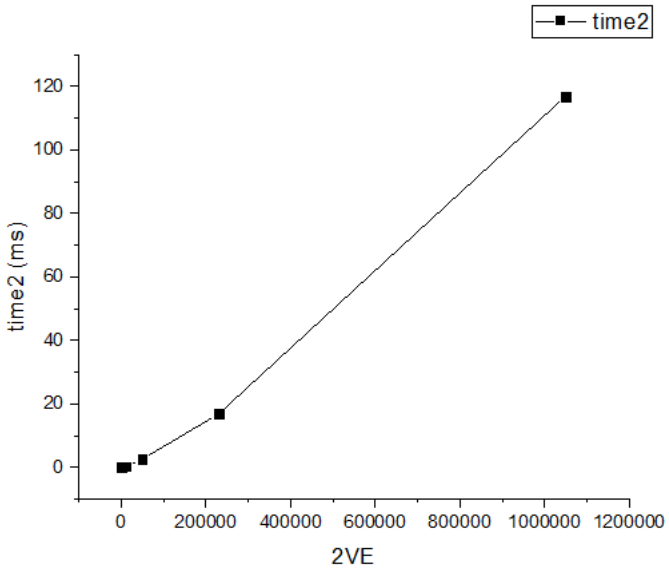
2. Johnson 算法的运行时间为 $O(V^2lgV+VE)$

这里 $E=VlgV$，故 $V^2lgV+VE=2VE$。

作以 time2 为纵坐标，2VE 为横坐标的折线图如下，并对其进行线性拟合得：

	Intercept		Slope		Statistics
	Value	Standard Error	Value	Standard Error	Adj. R-Square
time2	-2.27121	1.81084	1.12174E-4	4.12791E-6	0.99327

即 time2 = 1.12174E-4 ($V^2lgV+VE$) – 2.27121。R-square 非常接近 1，即线性拟合结果很好，time2 和 $V^2lgV+VE$ 约成线性关系，符合 time2=O($V^2lgV+VE$)的理论结果。



$V^2lgV+VE$	time2/ms
384	0.0080
2048	0.0573
10240	0.3700
49152	2.5927
229376	16.8173
1048576	116.8160

6. 实验心得

大部分情况下，边数为 VlogV 的图，都只有一个强连通分量。