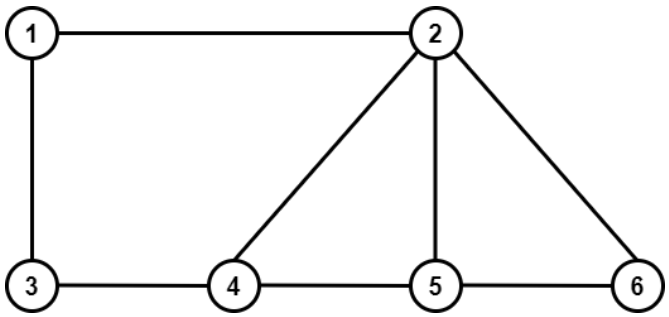
CSE 101
数据结构和算法介绍 编程作业2

广度优先搜索和最短路径
本作业的目的是在C语言中实现一个图形ADT和一些相关的算法。这个项目将利用你在第一部分的List ADT。首先阅读关于图算法的讲义，以及附录B.4、B.5和文本中的22.1、22.2节。

图的邻接列表表示由一个列表数组组成。每个List对应于图中的一个顶点，并给出该顶点的邻居。例如，该图



有邻接列表表示

```
1: 2 3
2: 1 4 5 6
3: 1 4
4: 2 3 5
5: 2 4 6
6: 2 5
```

你将创建一个Graph
ADT，使用这种方法来表示图形。每个顶点将用1到*n*范围内的整数标签来标识，其中*n*是图中顶点的数量。该项目中的客户端程序将被称为`FindPath.c`，它将使用Graph
ADT来寻找顶点对之间的最短路径（即具有最少边的路径）。它将接受两个命令行参数，如下所示。

```
$ FindPath input_file output_file
```

文件格式
输入文件将分为两部分。第一部分是由一个整数*n*组成的行，表示图中顶点的数量。接下来的每一行将用一对1到*n*的不同数字表示一条边，用空格隔开。这些数字是相应边的终端顶点。输入文件的第一部分定义了图形，并将由一个包含 "0 0 "的假行来结束。读取这些行后，你的程序将把图形的邻接列表表示法打印到输出文件中。例如，下面几行定义了上图中的图形，并导致打印出上述邻接列表的表示。

```
6
1 2
1 3
2 4
2 5
2 6
3 4
4 5
5 6
0 0
```

输入文件的第二部分将由若干行组成，每行由一对1-*n*范围内的整数组成，中间用空格分隔。每行指定图形中的一对顶点；一个起点（源）和一个终点（目的）。输入的第二部分也将由虚线 "0 0 "来结束。对于每一对源-目的地，你的程序将做以下工作。

- 从给定的源顶点开始执行广度优先搜索（BFS）。这将为图中的每个顶点分配一个父顶点（也称为*前身*，可能为零）。BFS算法将在课堂上讨论，并在下面作一般性描述。BFS的伪代码可以在课文的第22.2节中找到，也可以在这里的课堂网页上看到。

- 使用BFS的结果来打印出源顶点到目的顶点的距离，然后使用前人的结果来递归打印出从源到目的的最短路径。见文中第22.2节中的算法Print-Path，也见这里。

例子

输入文件

```
：6
1 2
1 3
2 4
2 5
2 6
3 4
4 5
5 6
0 0
1 5
3 6
2 3
4 4
0 0
```

输出文件。

```
1: 2 3
2: 1 4 5 6
3: 1 4
4: 2 3 5
5: 2 4 6
6: 2 5

与1的距离                到5是 2
一个最短的1-5路径        是。1 2 5

从3的距离                到6是 3
一个最短的3-6路径        是。3 1 2 6

与2的距离                到3是 2
一个最短的2-3条路径      是：2 1 3

从4开始的距离 一个最短   到4是    0
的4-4路径                是。4
```

如果没有从源头到目的地的路径（如果图是断开的，可能会发生这种情况），那么你的程序将打印一条这样的信息。请注意，连接一对给定顶点的最短路径可能不止一条。BFS发现的特定路径取决于它在每个邻接列表中通过顶点的顺序。在这个项目中，我们采用的惯例是顶点总是按照排序的顺序来处理，即按照顶点标签的增加来处理。BFS的输出由以下因素唯一决定

这个要求。因此，你的Graph
ADT应该按照排序的顺序维护邻接列表。下面的例子代表了一个断开连接的图。

输入文件 。	输出文件。
7	1: 4 5
1 4	2: 3 6
1 5	3: 2 7
4 5	4: 1 5
2 3	5: 1 4
2 6	6: 2 7
3 7	7: 3 6
6 7	
0 0	与2的距离 到7是 2
2 7	一个最短的2-7路径 是：2 3 7
3 6	
1 7	从3的距离 到6是 2
0 0	一个最短的3-6路径 是。3 2 6
	从1号1-7号路径存在的距离 到7是 宇宙空间

你的程序的操作可以分成两个基本步骤，与两组输入数据相对应。

1. 读取并存储图形，并打印出其邻接列表表示。

2. 输入一个循环，处理输入的第二部分。循环的每一次迭代都应该读入一对顶点（源点、目的地），在源点上运行BFS，打印到目的地顶点的距离，然后找到并打印所产生的最短路径（如果存在），或者打印一条从源点到目的地不存在路径的信息（如上例）。

什么是广度优先搜索？给定一个图*G*和一个顶点*s*，称为*源*顶点，BFS系统地探索*G*的边，以发现每一个从*s*可以到达的顶点，它计算从*s*到所有这些可到达顶点的距离。对于任何从*s*到达的*顶点*，BFS树中从*s*到*v*的唯一路径是*G*中从*s*到*v*的最短路径。Breadth First Search之所以被称为BFS，是因为它在边界的广度上均匀地扩展已发现和未发现的顶点之间的边界；也就是说，该算法在发现距离*k+1*的任何顶点之前发现了所有距离*s*的顶点。为了跟踪其进展并构建树，BFS要求*G*中的每个顶点*v*拥有以下属性：一个颜色*color*[*v*]，可以是白色、灰色或黑色；一个距离*d*[*v*]，是源*s*到顶点*v*的距离；一个父（或前辈）*p*[*v*]，是指*v*在BFS树中的父。在BFS执行过程中的任何时候，白色顶点是那些尚未被发现的顶点，黑色顶点已经完成，灰色顶点已经被发现，但不是所有的邻居都被发现。因此，灰色顶点形成了未发现的顶点和已完成的顶点之间的边界。BFS使用一个FIFO队列来管理灰色顶点的集合。使用pa2的List
ADT来实现这个FIFO队列，以及代表图形本身的邻接列表。

`Graph.c`定义了一个名为`GraphObj`的结构，`Graph.h`将定义一个名为`Graph`的类型，它是指向这个结构的指针。(在这一点上，重新阅读《C语言中的ADT和模块》讲义是个好主意）。在不进一步了解BFS的细节的情况下，我们可以看到`GraphObj`结构中需要以下字段。

- 一个Lists数组，其i^{th}元素包含顶点i的邻居。
- 一个ints（或字符，或字符串）数组，其i^{th}元素是顶点i的颜色（白、灰、黑）。
- 一个ints数组，其i^{th}元素是顶点i的父节点。
- 一个ints数组，其i^{th}元素是（最近的）来源到顶点i的距离。

您还应该包括存储顶点数量（称为图的*顺序*），边的数量（称为图的*大小*），以及最近被用作BFS源的顶点的标签的字段。建议所有数组的长度为$n+1$，其中n为图中顶点的数量，并且只使用索引1到n。这样一来，数组索引就可以直接与顶点标签相识别。

你的Graph ADT需要通过Graph.h文件导出以下操作。

```
/***构造器-破坏器 ***/ Graph
newGraph(int n);
void freeGraph(Graph* pG)。

/****访问函数 ***/ int
getOrder(Graph G);
int getSize(Graph G);
int getSource(Graph G);
int getParent(Graph G, int u);
int getDist(Graph G, int u)。
void getPath(List L, Graph G, int u)。

/***操纵程序 ***/ void makeNull(Graph
G);
void addEdge(Graph G, int u, int v);
void addArc(Graph G, int u, int v);
void BFS(Graph G, int s)。

/**** 其他业务 ****/
void printGraph(FILE* out, Graph G);
```

除了上述原型之外，Graph.h还将定义Graph类型以及#define常量宏INF和NIL，分别代表无穷大和未定义的顶点标签。为了实现BFS的目的，任何负的int值都是INF的适当选择，任何非正的int都可以代替NIL，因为所有有效的顶点标签都是正整数。INF和NIL当然应该是不同的整数。

函数newGraph()返回一个Graph，指向一个新创建的GraphObj，代表一个有n个顶点和没有边的图形。函数freeGraph()释放了与Graph *pG相关的所有堆内存，然后将句柄*pG设置为NULL。函数getOrder()和getSize()返回相应的字段值，getSource()返回最近在函数BFS()中使用的源顶点，如果BFS()还没有被调用，则返回NIL。函数getParent()将返回由BFS()创建的BFS树中顶点u的父级，如果BFS()尚未被调用，则返回NIL。函数getDist()将返回从最近的BFS源到顶点u的距离，如果BFS()还没有被调用，则返回INF。函数getPath()将G中从源到u的最短路径的顶点添加到列表L中，如果不存在这样的路径，则将值NIL添加到L中。getPath()有一个前提条件getSource(G)!=NIL，所以在调用getPath()之前必须调用BFS()。函数getParent()、getDist()和getPath()都有前提条件$1 \leq u \leq \text{getOrder}(G)$。函数makeNull()删除了$G$的所有边，将其恢复到其

原始（无边）状态。(这在图论文献中被称为*空图*）。函数`addEdge()`插入一条连接*u*和*v*的新边，即*u*被添加到*v*的邻接列表中，*v*被添加到*u*的邻接列表中，你的程序需要按照标签增加的排序来维护这些列表。函数`addArc()`从*u*到*v*插入一条新的有向边，即*v*被添加到*u*的邻接列表中（但不是*u*到*v*的邻接列表中）。`addEdge()`和`addArc()`都有一个前提条件，即它们的两个`int`参数必须位于1到`getOrder(G)`的范围内。函数`BFS()`在具有源*s*的*Graph*上运行BFS算法，相应地设置*G*的颜色、距离、父和源字段。最后，函数`printGraph()`将*G*的邻接列表表示法打印到`out`指向的文件中。这个表示法的格式应该与上面的例子相匹配，所以客户只需要调用`printGraph()`就可以了。

就像所有用C语言编写的ADT模块一样，你必须包括一个名为`GraphTest.c`的测试客户端，以单独测试你的Graph操作。请注意，由于Graph ADT包括一个具有List参数的操作（即`getPath()`），Graph的任何客户端也是List的客户端。由于这个原因，`Graph.h`文件应该`#include`头文件`List.h`。（关于使用`.h`文件的公认政策，请参见讲义*C头文件指南*）。如同在pa1中一样，你将写一个Makefile文件来创建名为FindPath的可执行二进制文件。在你的Makefile中包括一个清除工具，删除所有可执行二进制文件和中间的`.o`文件。网站上有一个Makefile，你可以根据你的需要进行修改。

因此，你将总共提交八个文件。

你写的`List.c`
你写的`List.h`
你写的`Graph.c`
`Graph.h`由你编写
`GraphTest.c`由你编写
`FindPath.c`由你编写
提供的`Makefile`，可根据你的需要修改。
`README` 一个提交的文件清单，以及对评分员的任何说明

你还会在/Examples/pa2中找到一个叫做`GraphClient.c`的文件，它使用BFS计算一些图论的数量。不要交出这个文件，但如果你愿意，可以在你自己的测试中使用它。它应该被认为是对我们的Graph ADT的一个弱测试，不能代替你自己的`GraphTest.c`。

请尽早开始这个项目，并根据需要从我、课程导师和助教那里获得帮助。