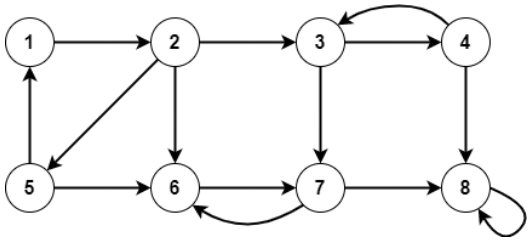
**CSE 101**
**数据结构和算法介绍 编程作业3**

在这项作业中，你将用C语言建立一个图形模块，实现深度优先搜索（DFS）算法。你将使用你的图形模块来寻找一个二维图形的强连接部分。阅读关于图算法的讲义，以及课文的22.3-22.5节。也请看课堂网页上的伪代码：Examples/Pseudo-Code/GraphAlgorithms。

一个二维图$G=(V, E)$被称为*强连接*，如果对于每一对顶点$u$，$v \in V$，顶点$u$可以从$v$到达，而顶点$v$可以从$u$到达，大多数有向图都不是强连接。一般来说，我们说一个子集$X \subseteq V$是*强连接的*，如果$X$中的每个顶点都能从$X$中的其他顶点到达，那么这个强连接的子集在*这个*属性上是最大的，称为$G$的强*连接部分*。
(i) $X$是强连接的，(ii) 在$X$上再加一个顶点会产生一个非强连接的子集。

**例子**

$G$




我们可以看到，这个数字图包含4个强连接部分。$C_1 = \{1, 2, 5\}, C_2 = \{3, 4\}, C_3 = \{6, 7\}$，以及 $C_4 = \{8\}$。

要找到一个数字图$G$的强成分，请调用DFS($G$)。当顶点完成后，将它们放在一个堆栈中。当DFS完成后，堆栈将存储按完成时间递减排序的顶点。接下来，计算$G$的转置$G^T$ 。$^T$(最后运行DFS($G^T$)，但在DFS的主循环中（第5-7行），按照第一次调用DFS时的完成时间递减的顺序处理顶点。这是通过从堆栈中弹出顶点来完成的。当整个过程完成后，产生的DFS森林中的树跨越了$G$的强连接部分。注意$G$的强连接部分与$G^T$ 的强连接部分是相同的。见文中第22.5节（第617页）中的强连接组件算法。

你的图模块将再次使用邻接列表表示。除其他事项外，它将提供运行DFS的能力，并计算有向图的转置。DFS要求顶点拥有颜色（白色、黑色、灰色）、发现时间、完成时间和父类的属性。下面是所需函数和原型的目录，构成Graph.h的大部分内容。

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

// 访问函数 int
getOrder(Graph G); int
getSize(Graph G);
int getParent(Graph G, int u); /*Pre: 1<=u<=n=getOrder(G) */
```

```
int getDiscover(Graph G, int u);   /*Pre: 1<=u<=n=getOrder(G)*/
int getFinish(Graph G, int u);   /* Pre: 1<=u<=n=getOrder(G) *
// 操纵程序
void addArc(Graph G, int u, int v); /* Pre: 1<=u<=n, 1<=v<=n */
void addEdge(Graph G, int u, int v); /* Pre: 1<=u<=n, 1<=v<=n*/
void DFS(Graph G, List S);    /* Pre: length(S)==getOrder(G) */

// 其他功能
Graph transpose(Graph G);
Graph copyGraph(Graph)。
void printGraph(FILE* out , Graph G);
```

函数newGraph()将返回一个包含$n$个顶点和无边的新图对象的引用。 freeGraph()释放所有与图相关的堆内存，并将其Graph参数设为NULL。函数getOrder()返回$G$中顶点的数量，而函数getParent()、getDiscover()和getFinish()返回给定顶点的适当字段值。请注意，顶点的父节点可能是NIL。在DFS被调用之前，顶点的发现和完成时间将是未定义的。你必须#define代表这些值的NIL和UNDEF的常量宏，并将定义放在Graph.h中。对函数addEdge()和addArc()的描述与pa2中完全一样。注意，和pa2中一样，要求邻接列表总是按照顶点标签的递增顺序进行处理。函数addEdge()和addArc()的责任是维持邻接列表的排序顺序。

函数DFS()将对$G$执行深度优先搜索算法。列表参数$S$在这个函数中有两个作用。首先，它定义了在DFS的主循环（5-7）中处理顶点的顺序。其次，当DFS完成时，它将按完成时间递减来存储顶点（因此$S$被认为是一个栈）。因此，$S$可以被归类为函数DFS()的输入和输出参数。DFS()有两个前提条件：(i) length($S$) == $n$，(ii) $S$包含一些整数{1, 2, … , $n$}的排列组合，其中$n$ = getOrder($G$)。您需要检查第一个前提条件，而不是第二个。

回顾一下DFS()调用递归算法Visit()（文中称为DFS-Visit()），并使用一个名为time的变量，该变量在对Visit()的所有递归调用中是静态的。请注意，这个函数在上述目录（Graph.h）中没有提及，因此应被视为Graph.c中的一个私有辅助函数。你可以将Visit()定义为Graph.c中的一个顶层函数，并让时间成为一个全局变量，其范围是整个文件。这种方法的缺点是Graph.c中的其他函数可以访问时间，并能够改变其值。由于这个原因，全局变量通常被认为是一种不好的编程做法。第二种方法是让时间成为DFS()中的一个局部变量，然后将时间的地址传递给Visit()，使其成为Visit()的输入输出变量。这也许是最简单的选择，并被推荐使用。第三种方法是再次让时间成为DFS()的一个局部变量，然后将Visit()的定义嵌套*到*DFS()的定义*中*。由于时间是DFS()的局部变量，它的范围包括Visit()的定义块，因此在对Visit()的所有递归调用中是静态的。如果你不习惯嵌套函数定义，这可能会很棘手，因为有范围问题需要处理。如果你选择了这个选项，首先要用几个例子做实验，以确保你知道它是如何工作的。请注意，尽管嵌套函数定义不是标准C语言的一个特征，也不被许多编译器所支持，但GNU gcc编译器*却*支持它（见https://gcc.gnu.org/onlinedocs/gcc/Nested-Functions.html）。实际上，还有第四种实现时间的方案，可能是所有方案中最简单的。只要给Visit()一个自己的本地时间副本，然后让它把时间的当前值作为输入，完成后返回时间的新值。没有全局变量，没有通过引用传递，也没有嵌套函数定义。这些设计决定是留给你的，而且是应该在你的README文件中说明的类型。

函数transpose()返回一个代表$G$的转置的新图对象的引用，copyGraph()返回一个作为$G$副本的新图的引用。printGraph()函数将$G$的邻接列表表示打印到out指向的文件中。很明显，这个项目中的图形模块和pa2中的模块有很多共同之处。如果你愿意，你可以简单地将本项目所需的功能添加到前一个项目中，尽管并不要求你这样做。你应该在你的README文件中记下这样的选择。

你的图形模块的客户端将被称为FindComponents。它将接受两个命令行参数，分别给出输入和输出文件的名称。

```
$ FindComponents infile outfile
```

FindComponents.c将做以下工作。

- 读取输入文件。
- 使用newGraph()和addArc()组装一个图对象$G$。
- 将$G$的邻接列表表示法打印到输出文件。
- 在$G$和$G$上运行DFS$^T$，在第二次调用中通过减少第一次调用的完成时间来处理顶点。
- 确定$G$的强成分。
- 将$G$的强成分按拓扑学排序打印到输出文件。

在第二次调用DFS()之后，可以用List参数$S$来确定$G$的强成分，并确定这些成分的拓扑排序。你应该在几个小例子上追踪强连接成分的算法（第617页），保持对List
$S$的跟踪，看看如何做到这一点。下面的例子说明了输入和输出文件的格式，它与本讲义第一页上的有向图相对应。

| 输入 | 输出。 |
|---|---|
| 。8 | G的邻接列表表示：1：2 |
| 1 2 | 2: 3 5 6 |
| 2 3 | 3: 4 7 |
| 2 5 | 4: 3 8 |
| 2 6 | 5: 1 6 |
| 3 4 | 6: 7 |
| 3 7 | 7: 6 8 |
| 4 3 | 8: 8 |
| 4 8 | |
| 5 1 | G包含4个强连接组件。组件1：1 5 2 |
| 5 6 | 组成部分2：3 4 |
| 6 7 | 组成部分3：7 6 |
| 7 6 | 组成部分4：8 |
| 7 8 | |
| 8 8 | |
| 0 0 | |

请注意，输入文件的格式与pa2的格式非常相似。第一行给出图中顶点的数量，随后几行指定有向边，输入以 "假 "行0 0结束。你需要提交以下八个文件。

README
Makefile
List.h
List.c
Graph.h
Graph.c
GraphTest.c
FindComponents.c

像往常一样，README包含了提交的文件目录和对评分者的任何特别说明。Makefile应该能够制作可执行文件GraphTest和FindComponents，并且应该包含一个清除所有二进制文件的工具。Graph.c和Graph.h分别是Graph模块的实现和接口文件。GraphTest.c 将包含你自己对 Graph模块的测试。FindComponents.c实现了本项目的顶级客户端和主程序。为了获得满分，你的项目必须实现所有要求的文件和函数，在编译时没有错误或警告，在单元测试中产生正确的输出，并且在valgrind下没有产生内存泄漏。现在大家都知道，如果忽略了所需的文件，拼错了任何文件名，以及提交了额外的不需要的文件，都会被扣分，但我还是要说：不要提交任何类型的二进制文件。

注意，FindComponents.c需要向DFS()函数传递一个List，所以FindComponents.c也是List模块的一个客户端。事实上，任何Graph的客户端都是List的客户端，只是因为DFS()中存在List参数。因此，Graph.h本身应该#include List.h文件。（关于这个问题，请参见题为 "*C头文件指南*"的讲义。）

这个项目的Makefile将被张贴在课程网页上，你可以根据自己的需要进行修改。像往常一样，尽早开始，如果有什么不完全清楚的地方，可以提问。