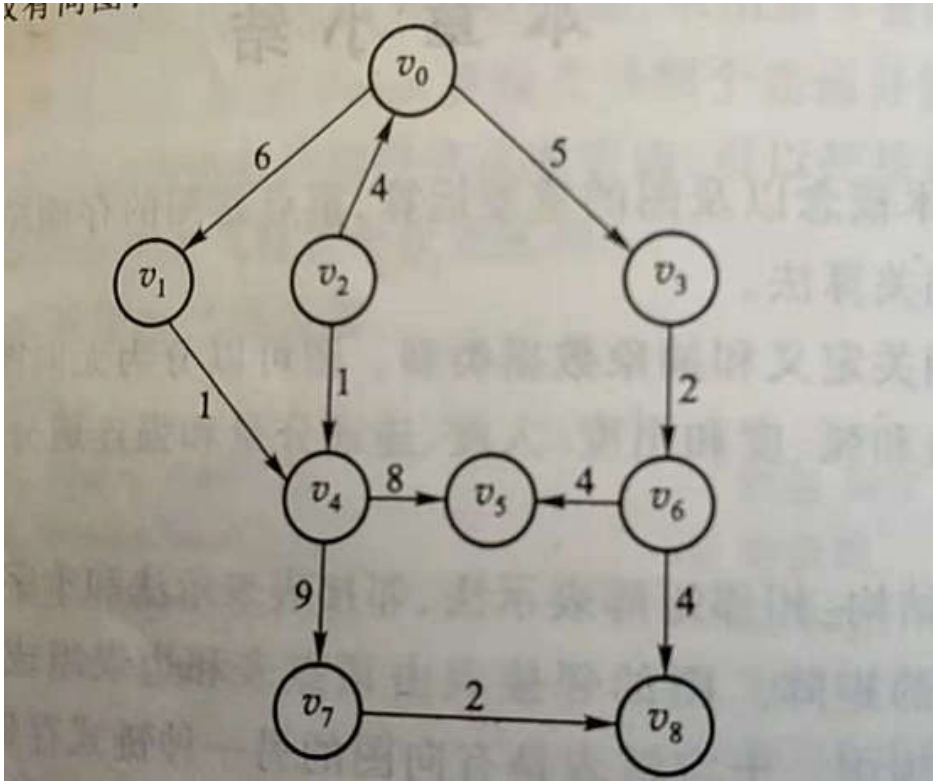
20181116 数据结构作业

1800022769 张靖昆 20181116

我承诺诚实作业，没有抄袭他人！



1．对于上图中的带权有向图：

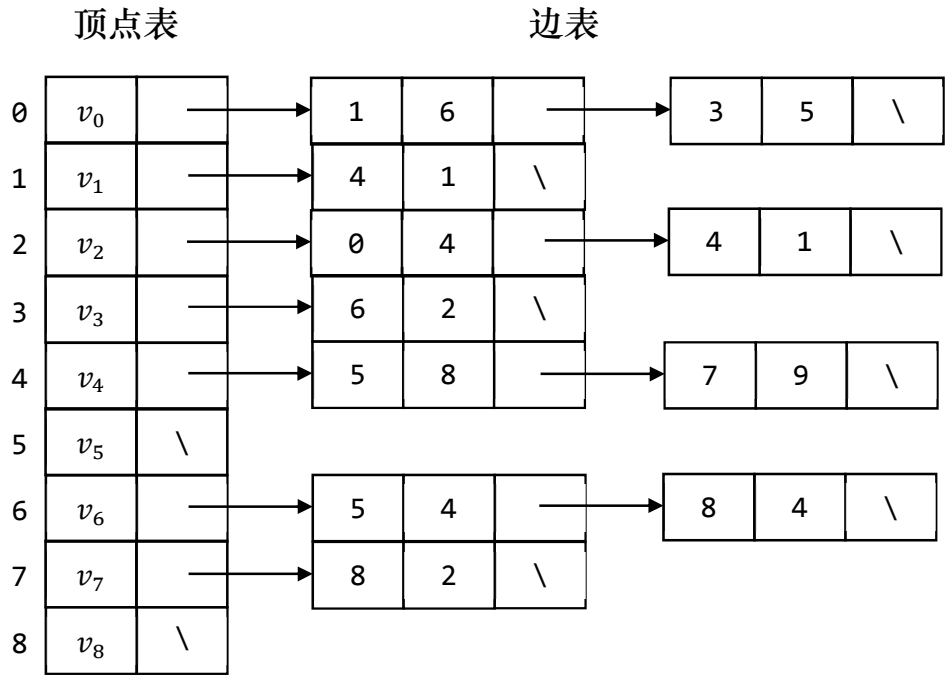
1）**写出其相邻矩阵**

从图中可知，共有 9 个顶点，从而建立 9×9 的矩阵，得到如下相邻矩阵：（由于 0 比较多，我将非 0 的部分全部以**深红加粗**醒目表示）

$$\begin{bmatrix} 0 & \mathbf{6} & \infty & \mathbf{5} & \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & \mathbf{1} & \infty & \infty & \infty & \infty \\ \mathbf{4} & \infty & 0 & \infty & \mathbf{1} & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty & \infty & \mathbf{2} & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & \mathbf{8} & \infty & \mathbf{9} & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \mathbf{4} & 0 & \infty & \mathbf{4} \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & \mathbf{2} \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

2）**画出其邻接表表示**

这里以出度表示。



3）**计算每个顶点的入度和出度**

出度和入度表如下：

顶点	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
出度	2	1	2	1	2	0	2	1	0
入度	1	1	0	1	2	2	1	1	2

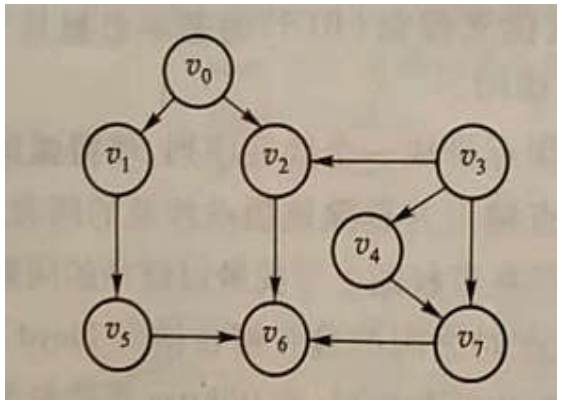
4）**如果每个指针需要 4 个字节，每个顶点的标号需要 2 个字节，每条边的权需要 2 个字节，则此图采用哪种表示法需要的空间较少**?

计算邻接矩阵的空间：由于邻接矩阵只记录权值，从而所占据空间为 9*9*2=162 字节；

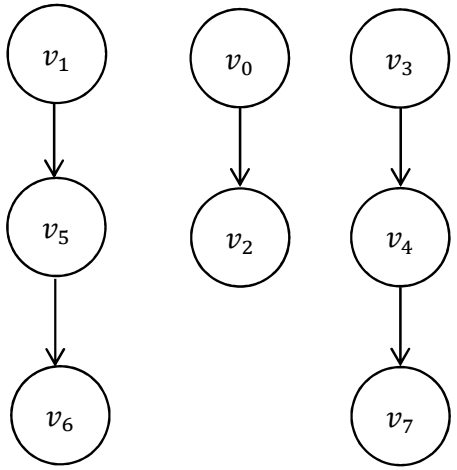
计算出度邻接表空间：顶点表中每个顶点占据 2+4=6 字节，从而 9 个顶点共 9*6=54 字节；边表中每个边结点占据 2+2+4=8 字节，从而 11 个边结点占据 11*8=88 字节，则邻接表所占总空间为 88+54=142 字节。

很显然，**邻接表占据空间较少**。

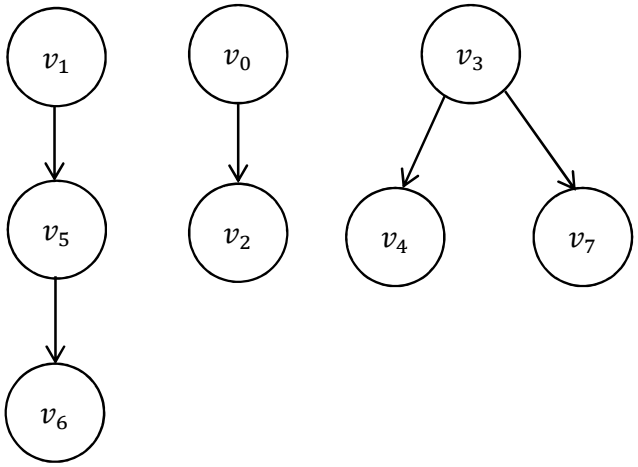
2．对于下面的有向图，从顶点v_1出发，分别画出其深度优先搜索和广度优先搜索生成的森林。



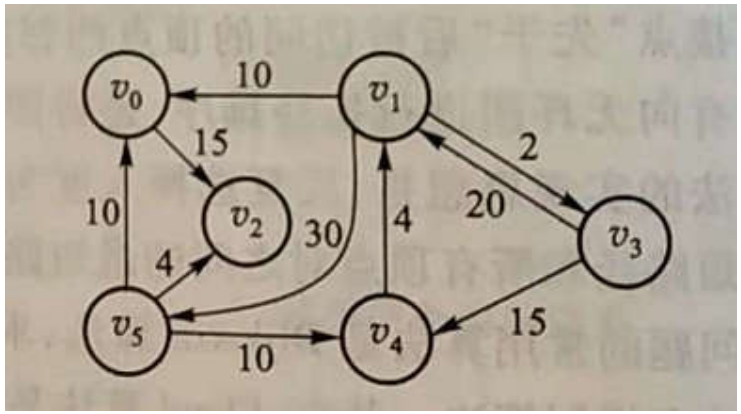
深度优先遍历生成的森林（其中一种）



广度优先遍历生成的森林（唯一一种）



3．求下面的有向图中从顶点v_4到其他顶点的全部最短路径及长度



1）**声明**:

a）设置长度为 6 的 path[]用于记录每个顶点最短路径中的前一个顶点的下标。

b）设置长度为 6 的 disc[]数组记录每个顶点最短路径的长度。

c）设置长度为 6 的 visit[]数组标记每个顶点是否被访问过。

d）设置顶点集合 vertexes[]，用于记录被加入最短路径区的顶点，主要是用于表示。

e）以邻接矩阵表示该有向图，记邻接矩阵为 weight[][]，weight[i][j]表示顶点 i 到顶点 j 的路径的权

值，为∞表示不可达。

f）**规则***：每次处理中，对于新加入的顶点，应当根据其所连接的顶点刷新数组中每个可达顶点的最短路径。即假设中间顶点为 k，则对于如果 k 对 i 顶点可达，且 disc[i] > disc[k]+weight[k][i]，则应当更新 disc[i] = disc[k]+weight[k][i]。

g）**规则****：对 path[]的处理规则为，经过选取权值最小的步骤后得到的新顶点 k，添加入 vertexes[]后，根据规则*应当对所有由顶点 k 可达的顶点的 disc[]进行更新，此时如若某个顶点 i 的 disc[]被更新，那么 path[i]=k。

2）**初始状态**：设置各数组的初值。

visit[]：令 visit[4]=1，其余全部为 0，代表初始状态下首顶点已被访问；

disc[]：搜索顶点v_4可达顶点的权值，令 disc[1]=4，其他全部为∞；

path[]：全部为-1。

下标	0	1	2	3	4	5

path[]	-1	-1	-1	-1	-1	-1
disc[]	∞	4	∞	∞	∞	∞
visit[]	0	0	0	0	1	0
vertexes[]	$\{v_4\}$					

3）**第一次搜索**：从 disc[]中且没被访问的顶点中选出最小的权值为 4，对应顶点为v_1。更新：

更新 visit[]：visit[1] = 1;

更新 disc[]：由顶点v_1可达的顶点有v_0、v_3、v_5，根据规则*更新 disc[0] = 14，disc[3] = 6，disc[5] = 34。

更新 path[]：根据规则**修改 path[]数组，得到 path[1] = 4，path[0] = 1，path[3] = 1，path[5] = 1;

下标	0	1	2	3	4	5
path[]	1	4	-1	1	-1	1

disc[]	14	4	∞	6	∞	34
visit[]	0	1	0	0	1	0
vertexes[]	$\{v_4, v_1\}$					

4）**第二次搜索**：从 disc[]中且没被访问的顶点中选出最小的权值为 6，对应顶点为v_3。更新：

更新 visit[]：visit[3] = 1;

更新 disc[]：由顶点v_3可达的顶点有v_1、v_4，根据规则*，发现 disc[1]＜disc[3]+weight[3][1]，又顶点v_4是出发顶点，从而不用更新；

更新 path[]：根据规则**，更新 path[3] = 1。

下标	0	1	2	3	4	5
path[]	1	4	-1	1	-1	1
disc[]	14	4	∞	6	∞	34
visit[]	0	1	0	1	1	0

vertexes[]	$\{v_4, v_1, v_3\}$

5）**第三次搜索**：从 disc[]中且没被访问的顶点中选出最小的权值为 14，对应顶点为v_0。更新：

更新 visit[]：visit[0] = 1;

更新 disc[]：由顶点v_0可达的顶点只有v_2，根据规则*，更新 disc[2] = 29。

更新 path[]：根据规则**，更新 path[2] = 0;

下标	0	1	2	3	4	5
path[]	1	4	0	1	-1	1
disc[]	14	4	29	6	∞	34
visit[]	1	1	0	1	1	0
vertexes[]	$\{v_4, v_1, v_3, v_0\}$					

6）**第四次搜索**：从 disc[]中且没被访问的顶点中选出最小的权值为 29，对应顶点为v_2。更新：

更新 visit[]：visit[2] = 1;

更新 disc[]：由于v_2出度为 0，从而 disc[]不用更新；

更新 path[]：由于 disc[]没有更新，从而 path[]不用更新。

下标	0	1	2	3	4	5
path[]	1	4	0	1	-1	1
disc[]	14	4	29	6	∞	34
visit[]	1	1	1	1	1	0
vertexes[]	$\{v_4, v_1, v_3, v_0, v_2\}$					

7）**第五次搜索**：从 disc[]中且没被访问的顶点中选出最小的权值为 34，对应顶点为v_5。更新：

更新 visit[]：visit[5] = 1;

更新 disc[]：由于从顶点v_5可达的顶点v_0、v_2、v_4，根据规则**，发现 disc[0]＜disc[5]+weight[5][0]，disc[2]＜disc[5]+weight[5][2]，又顶点v_4是源顶点，从而 disc[]不用更新。

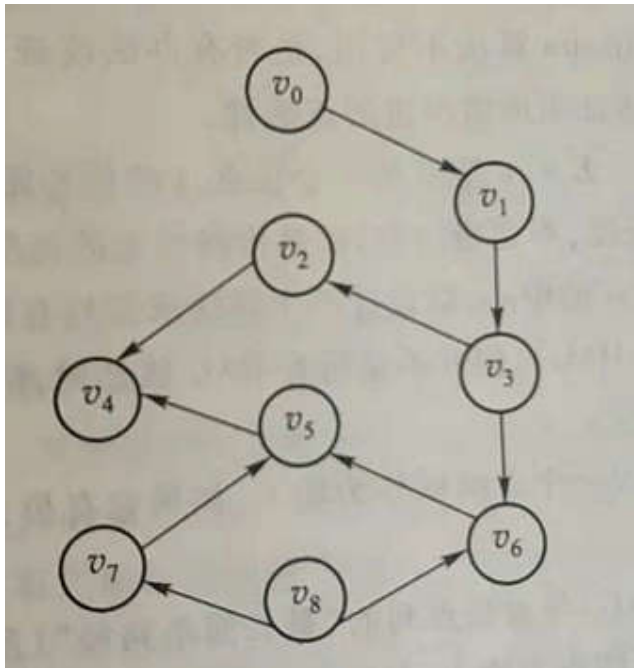
更新 path[]：由于 disc[]没有更新，从而不需要更新。

下标	0	1	2	3	4	5
path[]	1	4	0	1	-1	1
disc[]	14	4	29	6	∞	34
visit[]	1	1	1	1	1	1
vertexes[]	$\{v_4, v_1, v_3, v_0, v_2, v_5\}$					

经过上述查找过程，由顶点v_4到达其他顶点的最短路径以及长度如下表所示：

到达顶点	最短路径	长度
v_1	$v_4 \rightarrow v_1$	4
v_3	$v_4 \rightarrow v_1 \rightarrow v_3$	6
v_0	$v_4 \rightarrow v_1 \rightarrow v_0$	14
v_2	$v_4 \rightarrow v_1 \rightarrow v_0 \rightarrow v_2$	29
v_5	$v_4 \rightarrow v_1 \rightarrow v_5$	34

4．拓扑排序的结果不是唯一的，对于下面有向图中的顶点进行拓扑排序，能够得到多少个不同的拓扑序列。



拓扑排序不唯一的说明：设置一个队列，每次从顶点中找到一个入度为 0 的顶点，添加入队列，并删除该顶点所有出边，循环前述操作直至队列中已经存满所有顶点，得到的序列就是拓扑序列。**拓扑序列的不唯一就来源于当存在多个入度为 0 的顶点时，选取顺序不一样，排序结果就不唯一。**

下表中的“列”表示每个顶点在拓扑排序中的序列位置，一一讨论所有的情况。

0	1	2	3	4	5	6	7	8	序号
v_0	v_1	v_3	v_2	v_8	v_6	v_7	v_5	v_4	1
					v_7	v_6	v_5	v_4	2
			v_8	v_2	v_6	v_7	v_5	v_4	3
					v_7	v_6	v_5	v_4	4
				v_6	v_2	v_7	v_5	v_4	5
					v_7	v_2	v_5	v_4	6
					v_7	v_5	v_2	v_4	7
				v_7	v_2	v_6	v_5	v_4	8
					v_6	v_2	v_5	v_4	9
					v_6	v_5	v_2	v_4	10

v_0	v_1	v_8	v_3	v_2	v_6	v_7	v_5	v_4	11
					v_7	v_6	v_5	v_4	12
				v_6	v_2	v_7	v_5	v_4	13
					v_7	v_2	v_5	v_4	14
					v_7	v_5	v_2	v_4	15
				v_7	v_2	v_6	v_5	v_4	16
					v_6	v_5	v_2	v_4	17
					v_6	v_2	v_5	v_4	18
			v_7	v_3	v_2	v_6	v_5	v_4	19
					v_6	v_2	v_5	v_4	20
						v_5	v_2	v_4	21
			v_3	v_2	v_6	v_7	v_5	v_4	22

v_0	v_8	v_1			v_7	v_6	v_5	v_4	23
				v_6	v_2	v_7	v_5	v_4	24
					v_7	v_2	v_5	v_4	25
					v_7	v_5	v_2	v_4	26
				v_7	v_2	v_6	v_5	v_4	27
					v_6	v_5	v_2	v_4	28
					v_6	v_2	v_5	v_4	29
			v_7	v_3	v_6	v_2	v_5	v_4	30
						v_5	v_2	v_4	31
					v_2	v_6	v_5	v_4	32
		v_7	v_1	v_3	v_6	v_2	v_5	v_4	33
						v_5	v_2	v_4	34

					v_2	v_6	v_5	v_4	35
v_8	v_0	v_1	v_3	v_2	v_6	v_7	v_5	v_4	36
					v_7	v_6	v_5	v_4	37
				v_6	v_2	v_7	v_5	v_4	38
					v_7	v_2	v_5	v_4	39
						v_5	v_2	v_4	40
				v_7	v_2	v_6	v_5	v_4	41
					v_6	v_2	v_5	v_4	42
						v_5	v_2	v_4	43
			v_7	v_3	v_2	v_6	v_5	v_4	44
					v_6	v_2	v_5	v_4	45
						v_5	v_2	v_4	46

v_8		v_7	v_1	v_3	v_2	v_6	v_5	v_4	47
					v_6	v_2	v_5	v_4	48
						v_5	v_2	v_4	49
	v_7	v_0	v_1	v_3	v_2	v_6	v_5	v_4	50
					v_6	v_2	v_5	v_4	51
						v_5	v_2	v_4	52

非合并表格请查看一并提交的 EXCEL 表格，所有情况都在其中。总共 52 种。

5．第 9 题请看 DEV C++项目

6．**第 14 题**：证明：对于一个无向图 G = <V，E>，若 G 中各顶点的度均大于等于 2，则 G 中必有回路。

证明：利用数学归纳法进行证明，顶点数 n≥2：

1）当 n=2 时，易得唯一的两个顶点v_0和v_1度均为 2，即v_0和v_1的入度和出度均为 1，即v_0→v_1且v_1→v_0均可达，因此从v_0出发可以回到v_1，存在回路；

2）假设当 n=k 时，存在以下回路$v_i \to v_j \to v_k \to \ldots \to v_i$，假设路径中的不重复顶点数目为 m，则 m≤k；

3）n=k+1 时，新加入一个顶点v_x，则该顶点至少与图中 2 个顶点相连。对v_x与其他顶点的相连情况分为以下几中情况讨论：

a）顶点v_x与回路$v_i \to v_j \to v_k \to \ldots \to v_i$中任一顶点都不相连，那么此时图中仍存在回路，即$v_i \to v_j \to v_k \to \ldots \to v_i$；

b）顶点v_x的出边中有一部分或者全部与回路$v_i \to v_j \to v_k \to \ldots \to v_i$中的顶点相连，此时回路$v_i \to v_j \to v_k \to \ldots \to v_i$依旧存在，且由于顶点$v_x$只有出边与该回路中顶点相连，因此不会构成包含$v_x$的新回路，但依旧存在回路。

c）顶点v_x的入边中有一部分或者全部与回路$v_i \to v_j \to v_k \to \ldots \to v_i$中的顶点相连，此时回路$v_i \to v_j \to v_k \to \ldots \to v_i$依旧存在，且由于顶点$v_x$只有入边与该回路中顶点相连，因此不会构成包含$v_x$新回路，但依旧存在回路。

d）顶点v_x有一条出边和一条入边与回路$v_i \to v_j \to v_k \to \ldots \to v_i$中的任一顶点$v_y$相连，那么显然$v_x$可以代替$v_y$构成新回路$v_i \to v_j \to v_k \to \ldots v_x \to \ldots \to v_i$，并且不会破坏原回路，即依旧存在回路。

e）顶点v_x有一条出边与回路$v_i \to v_j \to v_k \to \ldots \to v_i$中的任一顶点$v_y$相连，有一条入边与与回路$v_i \to v_j \to v_k \to \ldots \to v_i$中的任一顶点$v_z$相连，那么可以构成新回路$v_i \to v_j \to v_k \to \ldots \to v_y \to v_x \to v_z \to \ldots \to v_i$，且原回路依然存在，

即依然存在回路。

f) 顶点v_x有多个出边和多个入边与回路$v_i \to v_j \to v_k \to \ldots \to v_i$中的多个顶点任意互连，由 d)和 e)可知，可以构成至少一个包含顶点v_x新回路，且原回路依旧存在，即依然存在回路。

综上，当 G 中各顶点的度均大于等于 2，则 G 中必有回路。

7．**第 19 题**：对于一个具有 n 个顶点和 e 条边的有向图 G = <V，E>，证明：求其强连通分量的算法所需的时间复杂度是 O(n + e)。

证明:

1) 首先证明确定一个顶点数为 n’和边数为 e’的图 G’=(V’,E’)是强连通分量的算法的时间复杂度为 O(n’ + e’)。

a) 设顶点集合 set(n)为由互相可达的顶点所组成，即对$\forall v_i$，$v_j \in$ set(n)，v_i到v_j有路径且v_j到v_i有路径。

b) 初始状态下加入顶点v_1进 set(n)，并记录顶点v_1已被访问。对v_1可达的任意顶点v_x，判断v_x是否可达v_1，如若可达且v_x不在集合 set(n)中，则加入集合 set(n)。可以看到加入集合 set(n)的顶点以及他们的出边都会被访问，而且没有加入集合 set(n)即v_1可达v_x但v_x不可达v_1的情况，v_1到v_x的边也会被访问，这

就使得即使两个顶点不是直接可达，那么他们之间的边也会被访问到。

c）再对新加入集合 set(n)的顶点做同样的操作，循环往复，直至最后一个顶点加入集合 set(n)；

d）由上述过程可以确定，求解过程中每个顶点、它们的所有出边以及出边所连顶点，都会被至少访问一次，易得求解算法的复杂度为 O(n’ + e’)。

2）假设有向图 G 中存在 m 个强连通分量也是 m 个子图，分别为$G_1 = \left(V_1，E_1\right)$，$G_2 = \left(V_2，E_2\right) \ldots G_m = (V_m，E_m)$，记它们的顶点数分别为$n_1, n_2 \ldots n_m$，记它们的和为$n_{sum} \leq$ n；记它们的边数分别为$e_1, e_2 \ldots e_m$，记它们的和为$e_{sum} \leq$ e；则求解每个强连通分量所用时间复杂度为 O$(n_1 + e_1)$，O$(n_2 + e_2)$…O$(n_m + e_m)$，那么总时间复杂度为

$$\sum_{i=1}^{m} O(n_i + e_i) = O\left(\sum_{i=1}^{m}(n_i + e_i)\right) = O\left(\sum_{i=1}^{m}(n_i) + \sum_{i=1}^{m}(e_i)\right) = O(n_{sum} + e_{sum})$$

3）再记剩余的顶点和边构成子图$G_r = \left(V_r，E_r\right)$，其顶点数为$n_r$，其边数为$e_r$，易得$n_r + \sum_{i=1}^{m} n_i = n$，$e_r + \sum_{i=1}^{m} e_i = e$。由于求解算法并不能也没必要区分一个顶点是属于强连通分量集合$(G_1 \ldots G_m)$的某一个，还是属于G_r，因此求解算法对于子图G_r的求解策略同求解强连通分量，从而子图G_r中的所有顶点以及每个顶点的所

有出边都会被至少访问一次，从而对于子图G_r上算法所花费的时间复杂度为$O(n_r + e_r)$。

4）综上，求解强连通分量的算法所花费的时间复杂度为$O(n_{sum} + e_{sum}) + O(n_r + e_r) = O(n + e)$。命题得证。