数据结构-16周

范围：图的BFS、图的连通性、最小生成树、
拓扑排序、Dijkstra算法

张天戈

本周习题

1. 请给出从加权无向图中生成最小生成树的两种方法，请分别描述其算法思想，并给出各自的时间复杂度。

最小生成树的定义与性质

山东大学 SHANDONG UNIVERSITY

定义

一个连通图的生成树包含图的所有顶点，并且只含尽可能少的边。对于生成树来说，若砍去它的一条边，则会使生成树变成非连通图；若给它增加一条边，则会形成图中的一条回路。

对于一个带权连通无向图 $G=(V, E)$，生成树不同，每棵树的权（即树中所有边上的权值之和）也可能不同。设$\Re$为 G 的所有生成树的集合，若 T 为$\Re$中边的权值之和最小的那棵生成树，则 T 称为 G 的最小生成树（Minimum-Spanning-Tree, MST）。

性质

1）最小生成树不是唯一的，即最小生成树的树形不唯一，$\Re$中可能有多个最小生成树。当图 G 中的各边权值互不相等时，G 的最小生成树是唯一的；若无向连通图 G 的边数比顶点数少 1，即 G 本身是一棵树时，则 G 的最小生成树就是它本身。

2）最小生成树的边的权值之和总是唯一的，虽然最小生成树不唯一，但其对应的边的权值之和总是唯一的，而且是最小的。

3）最小生成树的边数为顶点数减 1。

如何构造最小生成树？通用解法（贪心）

构造最小生成树有多种算法，但大多数算法都利用了最小生成树的下列性质：假设 $G=(V, E)$ 是一个带权连通无向图，U 是顶点集 V 的一个非空子集。若(u, v)是一条具有最小权值的边，其中 $u \in U$，$v \in V-U$，则必存在一棵包含边(u, v)的最小生成树。

基于该性质的最小生成树算法主要有 Prim 算法和 Kruskal 算法，它们都基于贪心算法的策略。对这两种算法应主要掌握算法的本质含义和基本思想，并能够手工模拟算法的实现步骤。

下面介绍一个通用的最小生成树算法：

```
GENERIC_MST(G){
    T=NULL;
    while T 未形成一棵生成树;
        do 找到一条最小代价边(u,v)并且加入 T 后不会产生回路;
            T=T∪(u,v);
}
```

最小生成树——Prim算法

算法思想

Prim 算法构造最小生成树的过程如图 6.15 所示。初始时从图中任取一顶点（如顶点 1）加入树 T，此时树中只含有一个顶点，之后选择一个与当前 T 中顶点集合距离最近的顶点，并将该顶点和相应的边加入 T，每次操作后 T 中的顶点数和边数都增 1。以此类推，直至图中所有的顶点都并入 T，得到的 T 就是最小生成树。此时 T 中必然有 $n-1$ 条边。

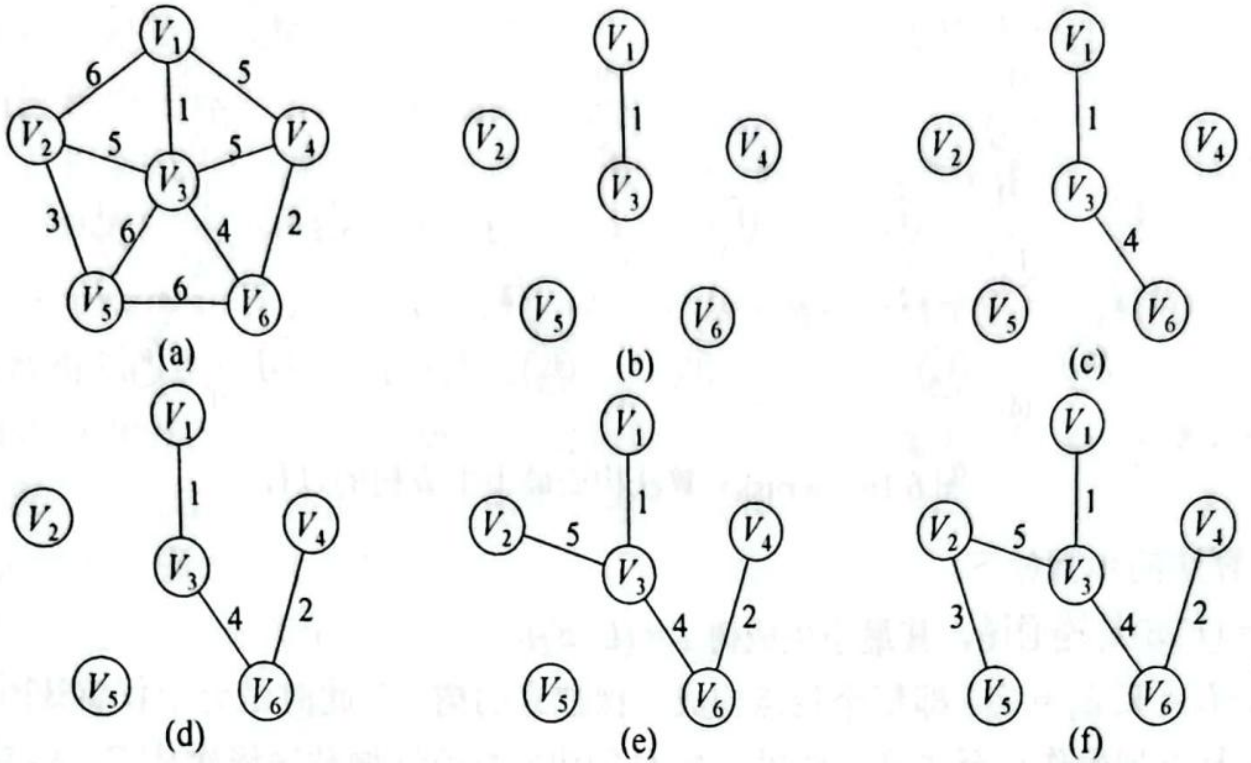


图 6.15　Prim 算法构造最小生成树的过程

最小生成树——Prim算法

算法伪代码与时间复杂度

Prim 算法的步骤如下：

假设 $G=\{V,E\}$是连通图，其最小生成树 $T=(U,E_T)$，E_T是最小生成树中边的集合。

初始化：向空树 $T=(U,E_T)$中添加图 $G=(V,E)$的任一顶点 u_0，使 $U=\{u_0\}$，$E_T=\varnothing$。

循环（重复下列操作直至 $U=V$）：从图 G 中选择满足$\{(u,v)|u\in U, v\in V-U\}$且具有最小权值的边$(u,v)$，加入树 T，置 $U=U\cup\{v\}$，$E_T=E_T\cup\{(u,v)\}$。

Prim 算法的简单实现如下：

```
void Prim(G,T){
    T=∅;                          //初始化空树
    U={w};                        //添加任一顶点 w
    while((V-U)!=∅){              //若树中不含全部顶点
        设(u,v)是使 u∈U 与 v∈(V-U),且权值最小的边;
        T=T∪{(u,v)};              //边归入树
        U=U∪{v};                  //顶点归入树
    }
}
```

Prim 算法的时间复杂度为 $O(|V|^2)$，不依赖于$|E|$，因此它适用于求解边稠密的图的最小生成树。

最小生成树——Kruskal算法

算法思想

Kruskal 算法构造最小生成树的过程如图 6.16 所示。初始时为只有 n 个顶点而无边的非连通图 $T=\{V,\{\}\}$，每个顶点自成一个连通分量，然后按照边的权值由小到大的顺序，不断选取当前未被选取过且权值最小的边，若该边依附的顶点落在 T 中不同的连通分量上，则将此边加入 T，否则舍弃此边而选择下一条权值最小的边。以此类推，直至 T 中所有顶点都在一个连通分量上。

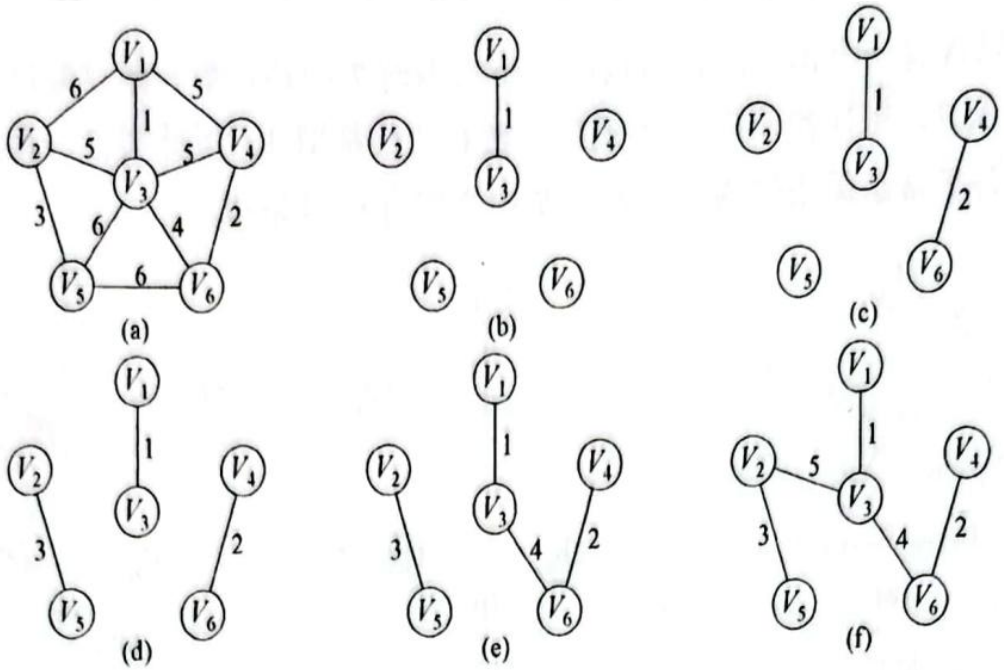


图 6.16　Kruskal 算法构造最小生成树的过程

最小生成树——Kruskal算法

山东大学 SHANDONG UNIVERSITY

算法伪代码与时间复杂度

Kruskal 算法的步骤如下：

假设 $G=(V,E)$是连通图，其最小生成树 $T=(U,E_T)$。

初始化：$U=V, E_T=\varnothing$。即每个顶点构成一棵独立的树，T 此时是一个仅含$|V|$个顶点的森林。

循环（重复下列操作直至 T 是一棵树）：按 G 的边的权值递增顺序依次从 $E-E_T$ 中选择一条边，若这条边加入 T 后不构成回路，则将其加入 E_T，否则舍弃，直到 E_T 中含有 $n-1$ 条边。

Kruskal 算法的简单实现如下：

```
void Kruskal(V,T){
    T=V;                          //初始化树 T，仅含顶点
    numS=n;                       //连通分量数
    while(numS>1){                //若连通分量数大于 1
        从 E 中取出权值最小的边(v,u);
        if(v 和 u 属于 T 中不同的连通分量){
        T=T∪{(v,u)};              //将此边加入生成树中
        numS--;                   //连通分量数减 1
        }
    }
}
```

根据图的相关性质，若一条边连接了两棵不同树中的顶点，则对这两棵树来说，它必定是连通的，将这条边加入森林中，完成两棵树的合并，直到整个森林合并成一棵树。

通常在 Kruskal 算法中，采用堆（见第 7 章）来存放边的集合，因此每次选择最小权值的边只需 $O(\log|E|)$ 的时间。此外，由于生成树 T 中的所有边可视为一个等价类，因此每次添加新的边的过程类似于求解等价类的过程，由此可以采用并查集的数据结构来描述 T，从而构造 T 的时间复杂度为 $O(|E|\log|E|)$。因此，Kruskal 算法适合于边稀疏而顶点较多的图。

拓扑排序定义

拓扑排序：在图论中，由一个有向无环图的顶点组成的序列，当且仅当满足下列条件时，称为该图的一个拓扑排序：

① 每个顶点出现且只出现一次。

② 若顶点 A 在序列中排在顶点 B 的前面，则在图中不存在从顶点 B 到顶点 A 的路径。

最短路径问题及解法

6.3 节所述的广度优先搜索查找最短路径只是对无权图而言的。当图是带权图时，把从一个顶点 v_0 到图中其余任意一个顶点 v_i 的一条路径（可能不止一条）所经过边上的权值之和，定义为该路径的带权路径长度，把带权路径长度最短的那条路径称为最短路径。

求解最短路径的算法通常都依赖于一种性质，即两点之间的最短路径也包含了路径上其他顶点间的最短路径。带权有向图 G 的最短路径问题一般可分为两类：一是单源最短路径，即求图中某一顶点到其他各顶点的最短路径，可通过经典的 Dijkstra（迪杰斯特拉）算法求解；二是求每对顶点间的最短路径，可通过 Floyd（弗洛伊德）算法来求解。

Dijkstra算法求单源最短路径问题

Dijkstra 算法设置一个集合 S 记录已求得的最短路径的顶点，初始时把源点 v_0 放入 S，集合 S 每并入一个新顶点 v_i，都要修改源点 v_0 到集合 V-S 中顶点当前的最短路径长度值（这里可能不太好理解？没关系，等下就会清楚）。

在构造的过程中还设置了两个辅助数组：

- dist[]：记录从源点 v_0 到其他各顶点当前的最短路径长度，它的初态为：若从 v_0 到 v_i 有弧，则 dist[i]为弧上的权值；否则置 dist[i]为∞。
- path[]：path[i]表示从源点到顶点 i 之间的最短路径的前驱结点。在算法结束时，可根据其值追溯得到源点 v_0 到顶点 v_i 的最短路径。

假设从顶点 0 出发，即 $v_0=0$，集合 S 最初只包含顶点 0，邻接矩阵 arcs 表示带权有向图，arcs[i][j]表示有向边<i,j>的权值，若不存在有向边<i,j>，则 arcs[i][j]为∞。

Dijkstra 算法的步骤如下（不考虑对 path[]的操作）：

1）初始化：集合 S 初始为{0}，dist[]的初始值 dist[i]=arcs[0][i]，i=1,2,…,n-1。

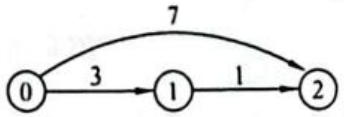
2）从顶点集合 $V-S$ 中选出 v_j，满足 dist[j]=Min{dist[i]|v_i∈V-S}，v_j 就是当前求得的一条从 v_0 出发的最短路径的终点，令 $S=S\cup\{j\}$。

3）修改从 v_0 出发到集合 $V-S$ 上任一顶点 v_k 可达的最短路径长度：若 dist[j]+arcs[j][k]<dist[k]，则更新 dist[k]=dist[j]+arcs[j][k]。

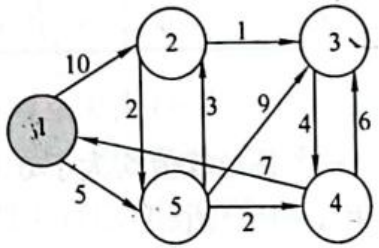
4）重复 2）～3）操作共 $n-1$ 次，直到所有的顶点都包含在 S 中。

Dijkstra算法求单源最短路径问题

步骤 3）也就是开头留下的疑问，每当一个顶点加入 S 后，可能需要修改源点 v_0 到集合 $V-S$ 中可达顶点当前的最短路径长度，下面举一简单例子证明。如下图所示，源点为 v_0，初始时 $S=\{v_0\}$，dist[1]=3，dist[2]=7，当将 v_1 并入集合 S 后，dist[2]需要更新为 4。



思考：Dijkatra 算法与 Prim 算法有何相似之处？



每轮得到的最短路径如下：
第1轮：1→5，路径距离为5
第2轮：1→5→4，路径距离为7
第3轮：1→5→2，路径距离为8
第4轮：1→5→2→3，路径距离为9

图 6.17　应用 Dijkstra 算法图

表 6.1　从 v_1 到各终点的 dist 值和最短路径的求解过程

顶点	第 1 轮	第 2 轮	第 3 轮	第 4 轮
2	10 $v_1\rightarrow v_2$	8 $v_1\rightarrow v_5\rightarrow v_2$	**8** $v_1\rightarrow v_5\rightarrow v_2$	
3	∞	14 $v_1\rightarrow v_5\rightarrow v_3$	13 $v_1\rightarrow v_5\rightarrow v_4\rightarrow v_3$	**9** $v_1\rightarrow v_5\rightarrow v_2\rightarrow v_3$
4	∞	**7** $v_1\rightarrow v_5\rightarrow v_4$		
5	**5** $v_1\rightarrow v_5$			
集合 S	{1, 5}	{1, 5, 4}	{1, 5, 4, 2}	{1, 5, 4, 2, 3}

1 2 3 4
0 10 ∞ ∞
8 14 7
8 13
9

0 8 9
-1 5 2

本周习题

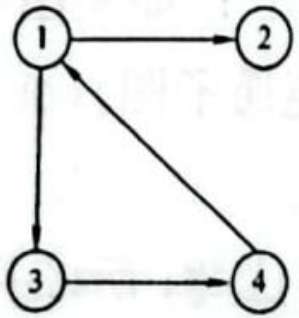
2. 下面是某有向加权图（顶点 A,B,C,D,E）的权重邻接矩阵，先给出一个拓扑序列，然后，使用 Dijkstra 算法依次计算出顶点 A 至其它各顶点的最短路径和最短路径长度。

	A	B	C	D	E
A		6		40	50
B				10	
C					20
D			30		10
E					

本周习题

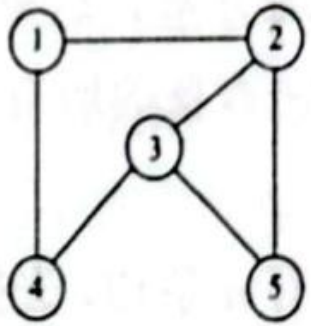
山东大学 SHANDONG UNIVERSITY

图的存储与基本操作——邻接矩阵



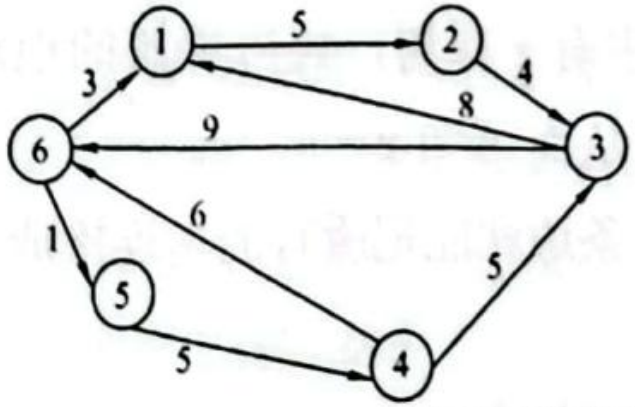
$$A_1=\begin{bmatrix}0&1&1&0\\0&0&0&0\\0&0&0&1\\1&0&0&0\end{bmatrix}$$

(a) 有向图G_1及其邻接矩阵



$$A_2=\begin{bmatrix}0&1&0&1&0\\1&0&1&0&1\\0&1&0&1&1\\1&0&1&0&0\\0&1&1&0&0\end{bmatrix}$$

(b) 无向图G_2及其邻接矩阵

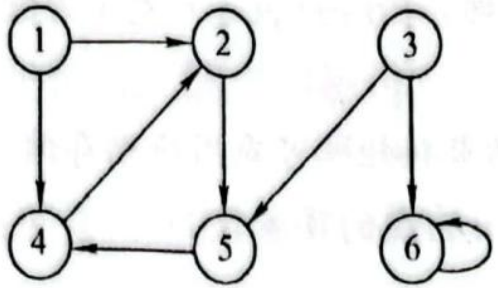


$$A_3=\begin{bmatrix}\infty&5&\infty&\infty&\infty&\infty\\\infty&\infty&4&\infty&\infty&\infty\\8&\infty&\infty&\infty&\infty&9\\\infty&\infty&5&\infty&\infty&6\\\infty&\infty&\infty&5&\infty&\infty\\3&\infty&\infty&\infty&1&\infty\end{bmatrix}$$

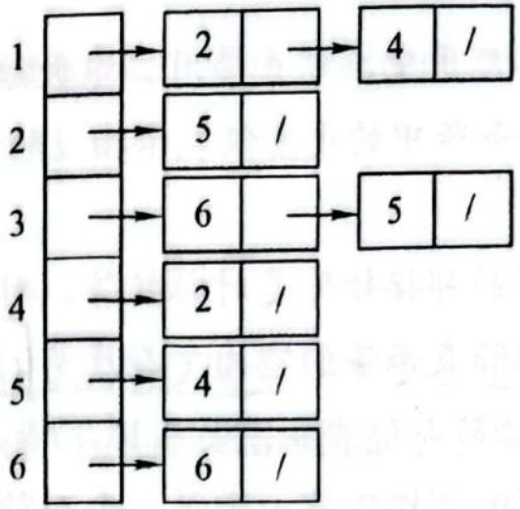
(c) 网及其邻接矩阵

图 6.5　有向图、无向图及网的邻接矩阵

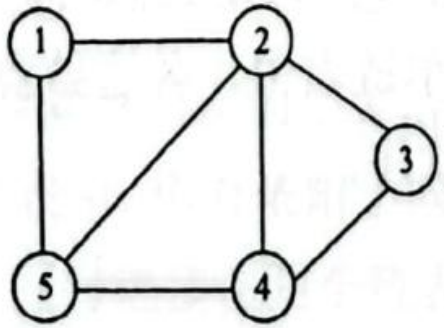
图的存储与基本操作——邻接链表



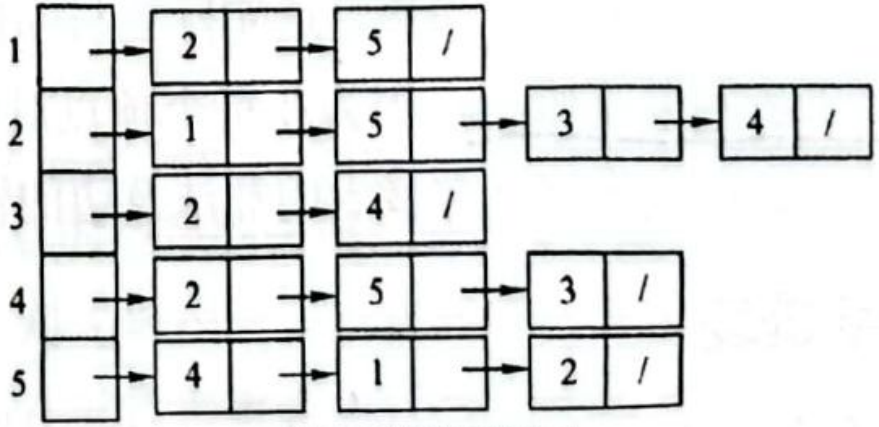
(a)有向图 G



(b)有向图 G 的邻接表的表示



(a)无向图G



(b)图G的邻接表的表示

本周习题

3. a 是一个(n-1)×n 的数组，用来描述一个 n 顶点图的邻接矩阵 A（如下图所示）。a 中没有描述矩阵的对角线。

（1）编写两个函数 Store 和 Retrieve 分别存储和搜索 A(i, j) 的值，每个函数的复杂性应为Θ（1）。

（2）编写函数 indegree（i），计算顶点 i 的入度，并分析其复杂度。

	1	2	3	4	5	6	7
1	1	1	1	0	0	0	0
2	1	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	1	1	0	0
5	0	0	0	0	1	1	1
6	0	0	0	0	1	1	1

本周习题

4. 请给出对 n 个顶点组成的无向图进行广度优先搜索遍历（BFS）的算法思想、伪代码，必要时给出注释，并分析算法的时空复杂度。

图的广度优先搜索遍历

换句话说，广度优先搜索遍历图的过程是以 v 为起始点，由近至远依次访问和 v 有路径相通且路径长度为 1, 2, …的顶点。广度优先搜索是一种分层的查找过程，每向前走一步可能访问一批顶点，不像深度优先搜索那样有往回退的情况，因此它不是一个递归的算法。为了实现逐层的访问，算法必须借助一个辅助队列，以记忆正在访问的顶点的下一层顶点。

广度优先搜索算法的伪代码如下：

```
bool visited[MAX_VERTEX_NUM];   //访问标记数组
void BFSTraverse(Graph G){      //对图 G 进行广度优先遍历
    for(i=0;i<G.vexnum;++i)
        visited[i]=FALSE;       //访问标记数组初始化
    InitQueue(Q);               //初始化辅助队列 Q
    for(i=0;i<G.vexnum;++i)     //从 0 号顶点开始遍历
        if(!visited[i])         //对每个连通分量调用一次 BFS
            BFS(G,i);           //v_i 未访问过，从 v_i 开始 BFS
}
void BFS(Graph G,int v){        //从顶点 v 出发，广度优先遍历图 G
    visit(v);                   //访问初始顶点 v
    visited[v]=TRUE;            //对 v 做已访问标记
    Enqueue(Q,v);               //顶点 v 入队列 Q
    while(!isEmpty(Q)){
        DeQueue(Q,v);           //顶点 v 出队列
        for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
                                //检测 v 所有邻接点
            if(!visited[w]){    //w 为 v 的尚未访问的邻接顶点
                visit(w);       //访问顶点 w
                visited[w]=TRUE;//对 w 做已访问标记
                EnQueue(Q,w);   //顶点 w 入队列
            }//if
    }//while
}
```

定义
算法思想
伪代码
时间复杂度分析

无论是邻接表还是邻接矩阵的存储方式，BFS 算法都需要借助一个辅助队列 Q，n 个顶点均需入队一次，在最坏的情况下，空间复杂度为 $O(|V|)$。

采用邻接表存储方式时，每个顶点均需搜索一次（或入队一次），故时间复杂度为 $O(|V|)$，在搜索任一顶点的邻接点时，每条边至少访问一次，故时间复杂度为 $O(|E|)$，算法总的时间复杂度为 $O(|V|+|E|)$。采用邻接矩阵存储方式时，查找每个顶点的邻接点所需的时间为 $O(|V|)$，故算法总的时间复杂度为 $O(|V|^2)$。

图的连通性

图的遍历算法可以用来判断图的连通性。

对于无向图来说，若无向图是连通的，则从任一结点出发，仅需一次遍历就能够访问图中的所有顶点；若无向图是非连通的，则从某一个顶点出发，一次遍历只能访问到该顶点所在连通分量的所有顶点，而对于图中其他连通分量的顶点，则无法通过这次遍历访问。对于有向图来说，若从初始点到图中的每个顶点都有路径，则能够访问到图中的所有顶点，否则不能访问到所有顶点。

故在 BFSTraverse() 或 DFSTraverse() 中添加了第二个 for 循环，再选取初始点，继续进行遍历，以防止一次无法遍历图的所有顶点。对于无向图，上述两个函数调用 BFS(G,i) 或 DFS(G,i) 的次数等于该图的连通分量数；而对于有向图则不是这样，因为一个连通的有向图分为强连通的和非强连通的，它的连通子图也分为强连通分量和非强连通分量，非强连通分量一次调用 BFS(G,i) 或 DFS(G,i) 无法访问到该连通分量的所有顶点，如图 6.14 所示。

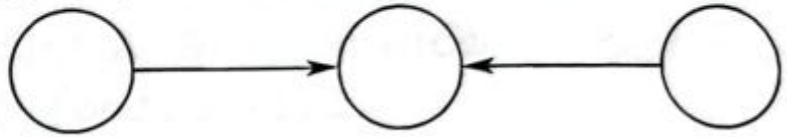


图 6.14　有向图的非强连通分量

本周习题

5. 请给出判断 n 个顶点的无向图是否为连通图的算法思想。

Q&A