研究生算法课课堂笔记

上课日期：2016 年 11 月 21 日　　第(1)节课

组长学号及姓名：1601214532 郑羽珊

组员学号及姓名：1601214511 江　月　1601111277 冯致远

一、 内容概要

1. 图的基本定义与应用（Basic definitions and applications）
2. 图的遍历（Graph traversal）
3. 二部图（Bipartite graphs）

二、 详细内容

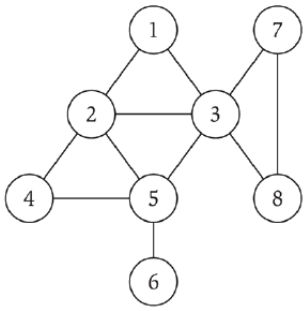
2.1 图的基本定义与应用（Basic definitions and applications）

2.1.1 图的定义

- **图：**图是由顶点的有穷非空集合和顶点之间边的集合组成，通常表示为：$G=(V,E)$

其中：G 表示一个图，V 是图G 中顶点的集合，E 是图G 中顶点之间边的集合。

— 表示图的大小的参数：顶点个数$n=|V|$，边的个数$m=|E|$



$V=\{1,2,3,4,5,6,7,8\}$

$E=\{$ 1-2, 1-3, 2-3, 2-4, 2-5, 3-5, 3-7, 3-8, 4-5, 5-6, 7-8 $\}$

$m=11, n=8$

图 2.1 图的定义示例

- **无向图：**若顶点之间的边没有方向，则称这条边为无向边，表示为(v_i,v_j) 。如果图的任意两个顶点之间的边都是无向边，则称该图为无向图。
- **有向图：**若顶点之间的边有方向，则称这条边为有向边，表示为$\langle v_i,v_j\rangle$。如果图的任意两个顶点之间的边都是有向边，则称该图为有向图。
- **无向完全图：**在无向图中，如果任意两个顶点之间都存在边，则称该图为无向完全图。

— 含有 n 个顶点的无向完全图有 $n\times(n-1)/2$ 条边。

- **有向完全图：**在有向图中，如果任意两个顶点之间都存在方向相反的两条边，则称该图为有向完全图。

  — 含有 n 个顶点的有向完全图有 $n\times(n-1)$ 条边。

2.1.2 图的表示

2.1.2.1 邻接矩阵（adjacency matrix）

- **基本思想**

用一个一维数组存储图中顶点的信息，用一个二维数组（称为邻接矩阵）存储图中各顶点之间的邻接关系

假设图 $G=(V,E)$ 有 n 个顶点，则邻接矩阵是一个 $n\times n$ 的方阵，定义为：

$$edges[i][j]=\begin{cases}1 & 若(v_i,v_j)\in E\\ 0 & 其他\end{cases}$$

公式 2.1 邻接矩阵定义

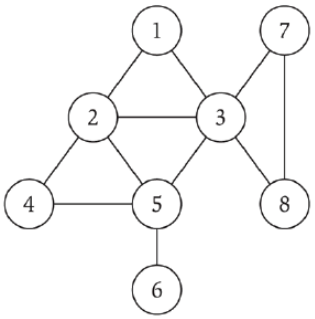
- **空间和时间复杂度**

  — 空间复杂度：$O(n^2)$

  — 时间复杂度

  ◆ 查找从结点 v_i 到结点 v_j 是否存在边——$O(1)$

  ◆ 遍历图中的所有边——$O(n^2)$



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

图 2.2 图的邻接矩阵表示

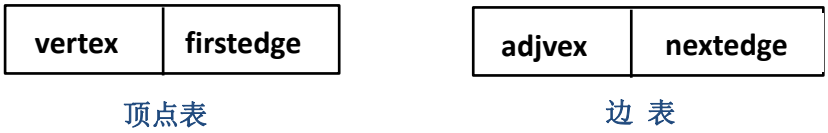
2.1.2.2 邻接表（adjacency lists）

- **基本思想**

对于图的每个顶点 u，将所有邻接于 u 的顶点链成一个单链表，称为顶点 u 的**边表**（对于有向图则称为出边表），所有边表的头指针和存储顶点信息的一维数组构成了**顶点表**。

- **结点结构**

邻接表有两种结点结构：顶点表结点和边表结点。



顶点表　　边 表

— vertex：数据域，存放顶点信息。

— firstedge：指针域，指向边表中第一个结点。

— adjvex：邻接点域，边的终点在顶点表中的下标。

— nextedge：指针域，指向边表中的下一个结点

- **空间和时间复杂度**

— 空间复杂度：$O(m+n)$

— 时间复杂度

◆ 查找从结点 v_i 到结点 v_j 是否存在边——$O(\deg ree(v_i))$

其中 degree 是指与 v_i 邻接的顶点个数。

◆ 遍历图中的所有边——$O(m+n)$

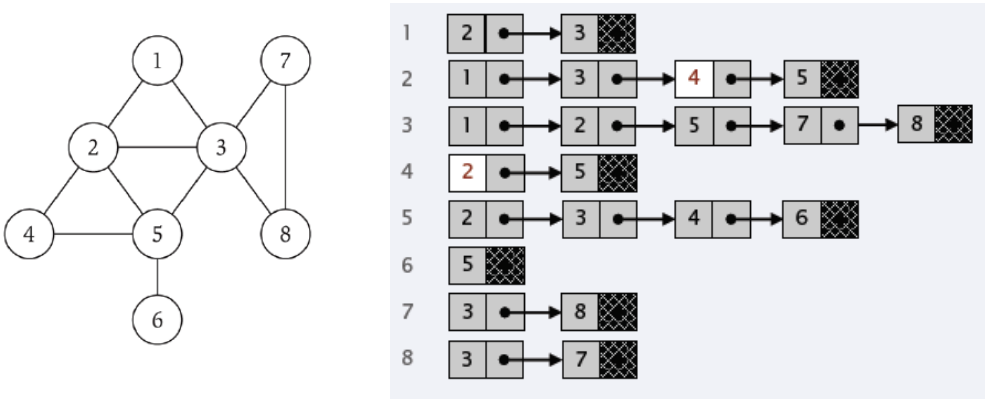


图 2. 3 图的邻接表表示

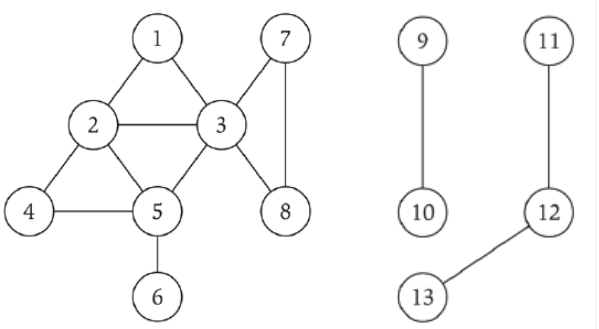
2.1.2.3 邻接矩阵与邻接表的比较

	空间性能	时间性能	适用范围	唯一性
邻接矩阵	$O(n^2)$	$O(n^2)$	稠密图	唯一
邻接表	$O(m+n)$	$O(m+n)$	稀疏图	不唯一

表 2.1 邻接矩阵与邻接表的比较

2.1.3 路径与回路

- **路径：**我们把无向图 $G=(V,E)$ 中的一条路径定义为具有如下性质的结点 $v_1,v_2,...,v_{k-1},v_k$ 的序列 P，其中每对连续的结点 v_i,v_{i+1} 被一条 G 中的边相交，P 常常被叫做从 v_1 到 v_k 的路径。
  - **路径长度**
    - 非带权图：路径上边的个数
    - 带权图：路径上各边的权之和
- **简单路径：**如果一条路径所有的结点都是互不相同的，就称为简单的。



cycle C = 1-2-4-5-3-1

图 2.4 环的示例

- **回路（环）：**第一个顶点和最后一个顶点相同的路径。
- **简单回路（简单环）：**除了第一个顶点和最后一个顶点外，其余顶点不重复出现的回路。

2.1.4 连通性

- **连通图：**在无向图中，如果从一个顶点 v_i 到另一个顶点 $v_j(i\neq j)$ 有路径，则称顶点 v_i 和 v_j 是连通的。如果图中任意两个顶点都是连通的，则称该图是连通图。
- **连通分量：**非连通图的极大连通子图称为连通分量。

- **强连通图：**在有向图中，对图中任意一对顶点 v_i 和 $v_j(i \neq j)$，若从顶点 v_i 到顶点 v_j 和从顶点 v_j 到顶点 v_i 均有路径，则称该有向图是强连通图。
- **强连通分量：**非强连通图的极大强连通子图。
- **生成树：**n 个顶点的连通图 G 的生成树是包含 G 中全部顶点的一个极小连通子图。
  - 含有 n-1 条边
  - 如果边的个数大于 n-1，则构成回路，少于 n-1，则不连通。
- **生成森林：**在非连通图中，由每个连通分量都可以得到一棵生成树，这些连通分量的生成树就组成了一个非连通图的生成森林。

2.1.5 树

- **树（Trees）：**一个无向图如果是连通的，且不包含一个环，我们就称它是一棵树。
  - 每棵 n 个结点的树恰好有 n-1 条边
- **根树（Rooted Trees）：**把根放在一个特点的结点 r 上，从 r 向外定向图的每条边；对于其他每个结点 v，我们认为 v 的父亲是在从 r 到它的路径上直接领先于 v 的节点 u。
  - 根树是一种层次的概念

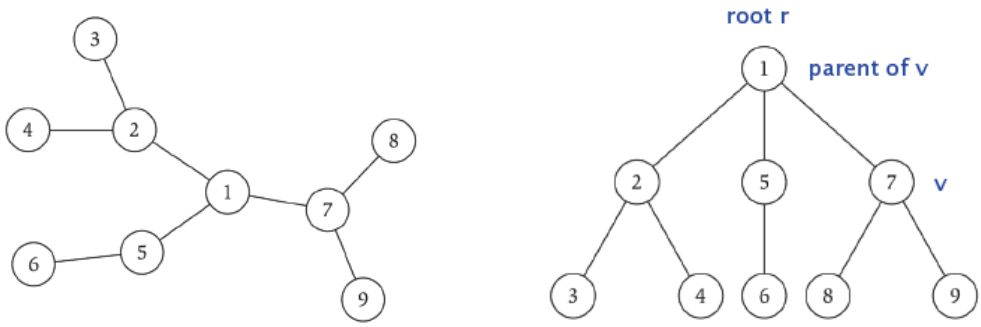


图 2. 5 同一棵树的两种画法，右图是根在结点 1 的画法

- **系统发生树（Phylogenetic tree）：**

系统发生树，又称为演化树（evolutionary tree），是表明被认为具有共同祖先的各物种间演化关系的树，常用它来描述物种之间的进化关系。在树中，每个节点代表其各分支的最近共同祖先，而节点间的线段长度对应演化距离（如估计的演化时间）。

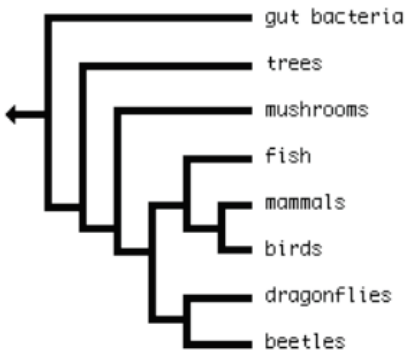


图 2.6 Phylogenetic tree 的示例

- **定理**

  假设 G 是一个具有 n 个结点的无向图。下面三个语句，已知任意两个可以推导出第三个。

  1） G 是连通图；

  2） G 不包含一个环；

  3） G 有 n-1 条边。

2.2 图的遍历（graph traversal）

2.2.1 宽度优先搜索（Breadth First Search）

2.2.1.1 基本概念

- **定义**

  已知图$G=(V,E)$和一个源顶点 s，宽度优先搜索从顶点 s 开始，辐射状地优先遍历其区域，每次增加一层，生成一棵根为 s 且包括所有可达顶点的宽度优先树。

- **基本思想**

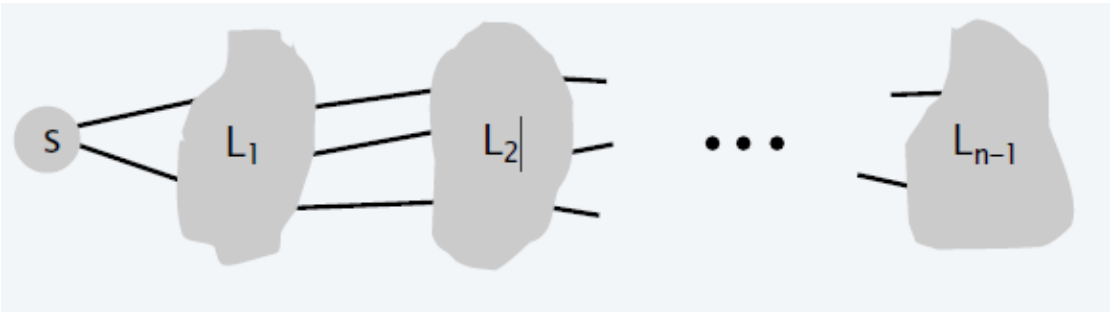


图 2.7 BFS 搜索示意图

1） 假设访问顶点 s；

2) 依次访问 s 的各个未被访问的邻接点 $s_1, s_2, \ldots, s_{k-1}, s_k$；

3) 分别从 $s_1, s_2, \ldots, s_{k-1}, s_k$ 出发依次访问它们未被访问的邻接点，并使“先被访问顶点的邻接点”先于“后被访问顶点的邻接点”被访问。直至图中所有与顶点 s 有路径相通的顶点都被访问到。

- **定理**

对每个 $j \geq 1$，由 BFS 产生的层 L，恰好由所有到 s 距离为 j 的结点组成，存在一条从 s 到 t 的路径当且仅当 t 出现在某个层中。

2.2.1.2 算法解析

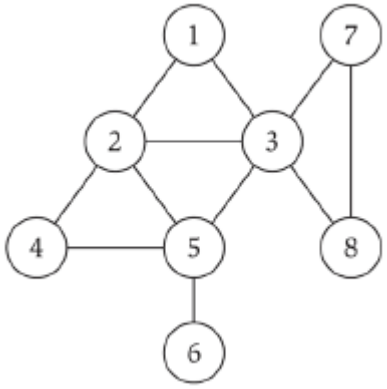


图 2.8 图 G 示例

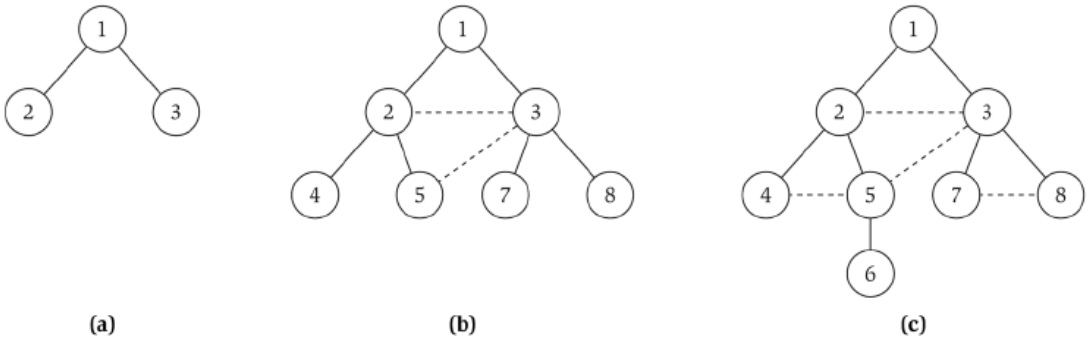


图 2.9 BFS 树 T 生成流程图

- **算法流程**

如图所示，图描述了一棵根在结点 1 的 BFS 树的结构，实线边是 BFS 树 T 的边，虚线边是 G 中不属于 T 的边，产生这棵树的流程如下：

1) 从结点 1 开始，层 L1 由结点 2,3 组成。

2) 然后通过按顺序（比如说先是 2，然后是 3）考虑在层 L1 中的结点而产生层 L2，于

是，只要我们看到 2，就发现结点 4 和 5,2 成为它们的父亲，当考虑结点 2 的时候，我们也发现一条通向 3 的边，但这条边并不加到 BFS 树中，因为我们已经知道了结点 3。当我们看结点 3 的时候首先发现 7 和 8，另一方面，3 到 5 的边是 G 中另一条不出现在树中的边，因为在见到来自结点 3 的这条边时，我们已经知道了结点 5。

3) 然后就是按次序考虑在层 L2 的结点，但是当我们检查到 L2 的时候，只有一个新结点被发现，就是结点 6，把它加到层 L3，边 $(4,5)$ 与 $(7,8)$ 不加到树中，因为他们没有导致新结点的发现。

4) 当检查结点 6 时没有发现新结点，因此没有结点放到层 L4，算法结束，图为整个 BFS 树。

- **定理**

设 T 是一棵宽度优先搜索树，设 x 和 y 是 T 中分别属于层 Li 和 Lj 的结点，并且设 (x, y) 是 G 的一条边，那么 i 和 j 至多差 1。

- **定理证明**

用反证法，假设 i 与 j 之间之差超过 1；特别地，设 i<j-1 。现在考虑 BFS 算法与 x 相交的边正在被检查到的一刻。因为 x 属于层 Li，那些从 X 发现的结点只能属于层 L_1+1，或者他们早被发现过；因此，如果 y 是 x 的一个邻居，那么它最迟应该在这个时刻被发现，因此应属于层 L_1+1 或者更早的层。

2.2.1.3 算法实现

- **宽度优先搜索的实现**

邻接表数据结构对于实现宽度优先搜索是理想的，维护一个长为 n 的数组 Discovered，作为判定该点是否被访问过，当扫描离开 u 的边来到边 (u,v) 时，以 v 是否被访问过来确定是否将 v 加入其中。同时表 L[i]表示与原点 s 距离为 i 的集合，伪代码如下：

BFS（s）：

置 Discovered[s]=true 且对所有其他的 v，置 Discovered[s]=false

初始化 L[0]由单个元素 s 构成

置层计数器 i=0

While L[i]不空

初始化一个空表 L[i+1]

For 每个结点 u $\in L[i]$

考虑每条关联到 u 的边 (u,v)

If Discovered[v]=false then

置 Discovered[v]=true

把边 (u,v) 加到树 T 上

把 v 加到表 L[i+1]

Endif

Endfor

把层计数器加 1

Endwhile

设图中边有 m 条，顶点有 n 个，整个算法用于考虑边的总时间为 $O(m)$，需要 $O(n)$ 的额外时间来建立表及管理数组 Discovered，则算法时间复杂度为 $O(m+n)$。

2.2.1.4 相关问题

- **引申小问题**

1） 判断最长路径（无权重）用什么方法比较好（BFS 或 DFS）？

答：BFS

2） 从根节点开始用 BFS 遍历，是否可能不生成原图中可能出现的最短路径树？

答：可能，图（b）是图（a）的最短路径树，但是用 BFS 无法生成。

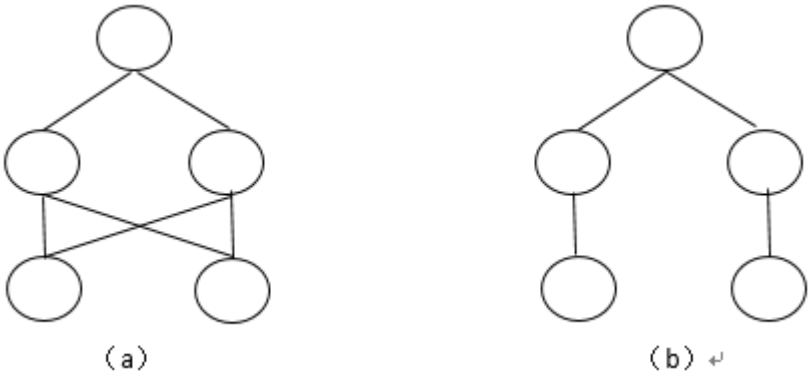


图 2.10 BFS 遍历不能生成的最短路径树示例

2.2.2 深度优先搜索（Depth First Search）

2.2.2.1 基本概念

- **定义**

  已知图$G=(V,E)$和一个源顶点 s，深度优先搜索从顶点 s 开始，从其中一条边出发，到达另一结点 v，再从 v 的一条边出发，以这种方式一直进行直至到达“终点”，然后回溯到未被探查的邻居结点重新开始，生成一棵根为 s 且包括所有可达顶点的深度优先树。

- **基本思想**

  1) 访问顶点 s；

  2) 依次访问 s 的各个未被访问的邻接点$s_1,s_2,...,s_{k-1},s_k$；

  3) 分别从$s_1,s_2,...,s_{k-1},s_k$出发依次访问它们未被访问的邻接点，并使“先被访问顶点的邻接点”先于“后被访问顶点的邻接点”被访问。直至图中所有与顶点 s 有路径相通的顶点都被访问到。

- **定理**

  对于给定的递归调用 DFS(u)，在这次激活和这个递归调用结束之间被标记为搜索过的所有结点都是 u 在 T 中的后代。

2.2.2.2 算法解析

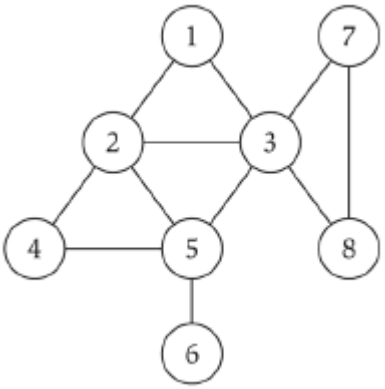


图 2.11 图 G 示例

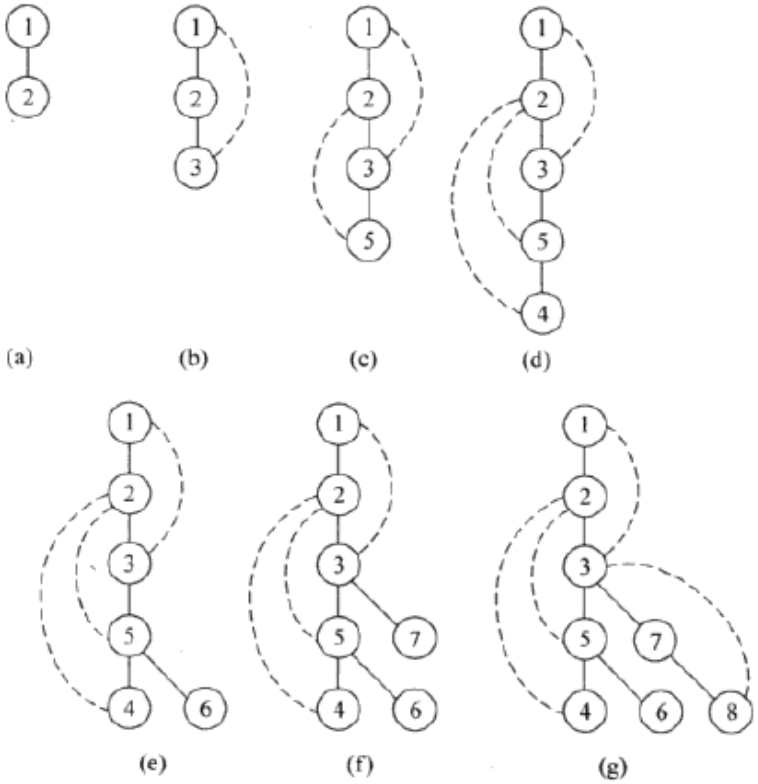


图 2.12 DFS 树 T 生成流程图

- **算法流程**

  如图所示，图描述了一棵根在结点 1 的 DFS 树的结构，实线边是 DFS 树 T 的边，虚线边是 G 中不属于 T 的边，产生这棵树的流程如下：

  1) 从结点 1 开始，由 DFS 的执行建立一条在 1,2,3,5,4 上的路径，在 4 时达到死终点。

  2) 由于已经到达死终点，找不到新的结点，因此它“回到”5，找到结点 6，再“回到”3 依次进行。

3) 当回到 3 结点找到 7 和 8，这时这个连通分支再也找不到新的结点了，因此递归一个个的终结，最后算法结束，图为整个 DFS 树。

- **定理**

设 T 是一棵深度优先搜索树，x 和 y 是 T 中结点，且 (x, y) 是 G 中不属于 T 的一条边，那么 x 或 y 之中一个是另一个的祖先。

- **定理证明**

假设 (x, y) 是 G 中的一条边，但不是 T 中的边，且不失一般性，假设 DFS 算法先到达 x，在 DFS（x）被执行期间，当边 (x, y) 被检查时，y 没有被加到 T 中是由于他被 DFS 调用了，因此它是在递归调用的激活和结束之间被发现的，因此 y 是 x 的一个后代。

2.2.2.3 算法实现

- **深度优先搜索的实现**

深度优先算法我们采用递归的方式实现，将所有与 u 邻接的结点加到一个将被考虑的节点表中，之后我们继续探查 u 的邻居 v，再把 v 的邻居加到我们正在维护的表中，我们按照栈的顺序进行探查，探查完 v 的邻居之后才能回到 u。同时使用一个 Explored 数组，当我们扫描 v 的关联边时才将 Explored[v]置为 true，伪代码如下：

```
DFS（s）:
    初始化 S 为具有一个元素 s 的栈
    While S 不空
        从 S 中取出一个节点 u
        If Explored[v]=false then
            置 Explored [v]=true
            For 每条与 u 关联的边（u，v）
                把 v 加到栈 S
            Endfor
        Endif
    Endwhile
```

设图中有 m 条，顶点有 n 个，如果图是由邻接表给出的，整个算法用于考虑加到 S 中的结点数的总时间为 $O(m)$，需要 $O(n)$ 的额外时间来建立表及管理数组 Explored，则算法时间复杂度为 $O(m+n)$。

2.3 二部图（bipartite graphs）

2.3.1 二部图定义及应用（The definition and application of bipartite graphs）

- **定义**

二部图分为左右两部分顶点，左边顶点只能和右边顶点相连，右边顶点只能和左边顶点相连，同一侧顶点不准有边相连，下图就是一个二部图。

- **应用**

典型的二部图应用场合有：稳定婚姻（Stable Marriage）、任务调度（Task Schedule）。

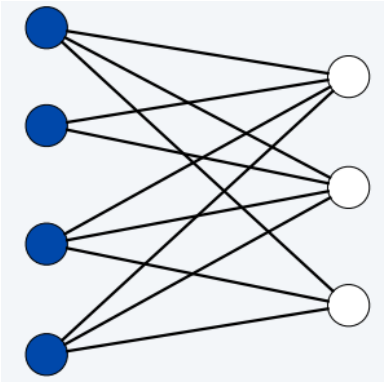


图 2.13 二部图示例

2.3.2 二部图性质（The attribute of bipartite graphs）

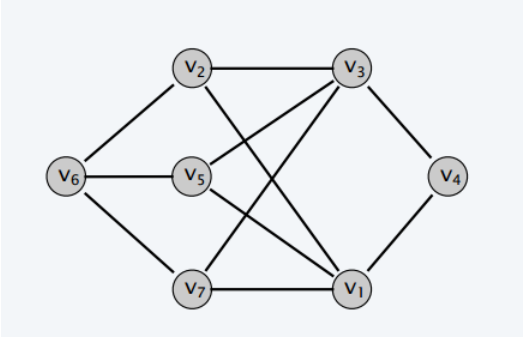


图 2.14 一个顶点不分排在两边的二部图

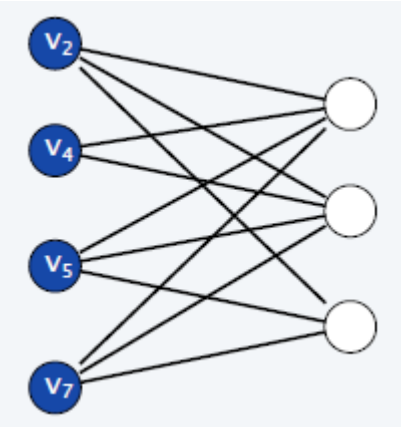


图 2. 15 图 2. 14 的另一种画法

测试一个图是否是二部图，用肉眼看不是好办法，因为有些图不一定把顶点分排在两边，如图 2.14 所示，但是图 2.14 与图 2.13 为同一二部图，具体画法如图 2.15 所示。

- **定理一**

  二部图中不能有长度为奇数的环（但可以有长度为偶数的环）

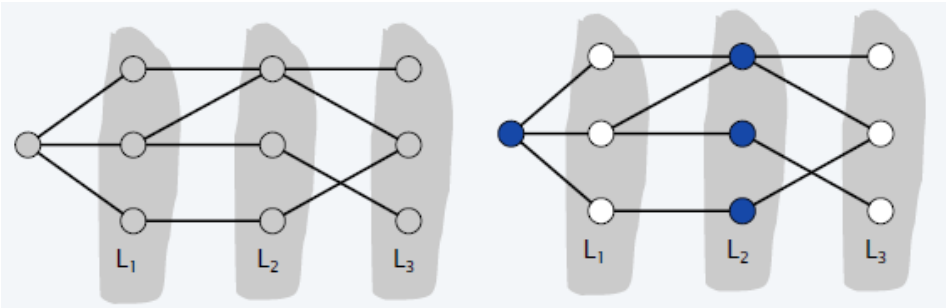
- **定理一证明**

  对二部图中的顶点用蓝白二色进行染色，则每条边的两个顶点颜色不同。如果有奇数环，是无法做到这一点的。

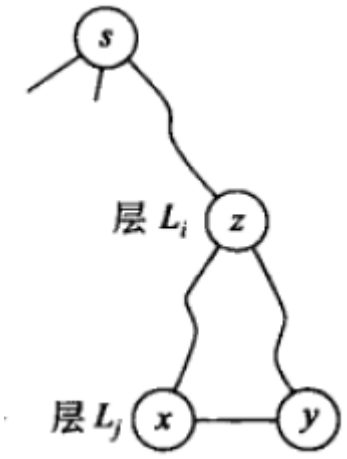
- **定理二**

  对于连通图 G，在由 BFS 产生的层中，同一层没有两个顶点间有边相连，则图为二部图，如有相连则其包含一个奇数环

- **定理二证明**



（a） （b）



（c）

图 2. 16 定理二证明示例图

假设同层之间点没有边，且图为连通图，根据 2.2.1.2 中的定理，两条边之间的层数最多差 1，则可以采用隔层涂色的方式填涂二部图，如图 2.16（b）所示，可以看没有一条边连接两个颜色相同的点，其为二部图。

假设 (x, y) 是在 Lj 层连通 x 和 y 的一条边，z 是 x 和 y 的最近公共祖先，假设 z 在 Li 层，则存在了一个 x→y→z→x 环，其长度为1+（j - i)+（j - i），其为奇数，所以是奇数环，由 2.3.2 的定理一可知，不是二部图。

参考文献

[1] Jon Kleinberg, Éva Tardos 著；张立昂，屈婉玲 译. 算法设计. 第一版， 北京：清华大学出版社，2007.3
[2] 王红梅，胡明，王涛 . 数据结构（C++版）. 第二版[M]，北京：清华大学出版社，2011.6
[3] 系统发生树的百度百科 http://baike.baidu.com/link?url=tlxRV6JG2X3EK6eFkVeFf4ClRDiuCAtVESDwYcGjXxrmp5hy4gQuJj5Wh782-kDaPhQ3G2IcaAj0ExuWfFOEyK
[4] Phylogenetic tree 的维基百科 https://en.wikipedia.org/wiki/Phylogenetic_tree
[5] Breadth First Search 的维基百科 https://en.wikipedia.org/wiki/Breadth-first_search
[6] Depth First Search 的维基百科 https://en.wikipedia.org/wiki/Depth-first_search