基本图算法

20.1 图的表示

对于图 G=(V,E)，有两种标准表示方法：邻接链表和邻接矩阵。它们每一个都可以表示有向图和无向图。
稀疏图：$|E| \ll |V|^2$。通常用邻接链表表示（更紧凑）。
稠密图： $|E|$与$|V|^2$相近。通常用邻接矩阵表示。

邻接链表

（确定顶点的度，稀疏图，图的遍历，大部分问题）

图 G=(V,E)的邻接链表表示由一个包含$|V|$条链表的数组Adj构成，每个顶点有一个链表。
对于每个顶点 $u \in V$， Adj [u] 包含所有与 u 有边相连的顶点 v，即 $(u,v) \in E$ 。

`ps：邻接表表示由一组邻接表组成，其中每个邻接表可以用链表，也可以用变长数组，散列表或红黑树等数据结构。

- 有向图：边 (u,v) 中的 v 记录在链表 Adj[u] 中。每条边记录一次，所有邻接表长度之和为 $|E|$ 。
- 无向图：边 (u,v) 中的 v 记录在 Adj[u] 中， u 记录在 Adj[v]中。每条边记录两次，所有邻接表长度之和为 $2|E|$ 。

邻接表表示需要的内存为 $\Theta(|V|+|E|)$
遍历所有边时间为 $\Theta(|V|+|E|)$ 。
若 $|E|=\Omega(|V|)$，则 $\Theta(|V|+|E|)=\Theta(|E|)$ 。

邻接链表也可以用来表示权重图，每条边有一个权重，权重函数为 $w: E \to R$， w(u,v) 表示边 (u,v) 的权重。
对于权重函数w(u,v)，直接将边(u,v)的权重值通过w(u,v)放到节点u的邻接链表中。

缺点：无法快速判断 (u,v) 是否是图中的边。只能在邻接链表中搜索结点v。
邻接矩阵表示克服了这个缺点，但需要更多的渐近存储。

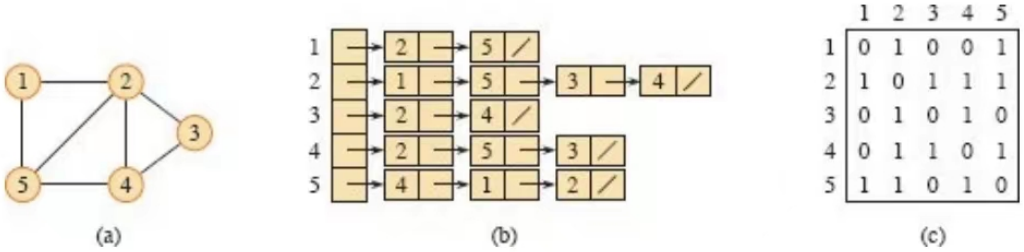


Figure 20.1 Two representations of an undirected graph. **(a)** An undirected graph G with 5 vertices and 7 edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

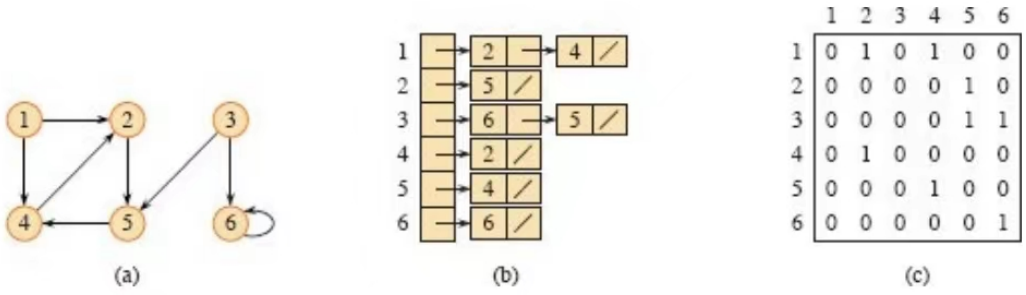


Figure 20.2 Two representations of a directed graph. **(a)** A directed graph G with 6 vertices and 8 edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

邻接矩阵

（边的存在与否，稠密图，边的插入删除）

将图中结点任意编号为1，2，…，|v|，形成|v|×|v|的矩阵A=（a_{ij}),满足以下条件：

$$aij = \begin{cases} 1 & (i,\ j) \in E \\ 0 & (i,\ j) \notin E \end{cases}$$

需要的内存: Θ($|V|^2$) 。

无向图的矩阵为一个对称矩阵，有 $A=A^T$ 。某些应用中，只需要存储上/下三角邻接矩阵。
ps:无向图可以存储上/下三角矩阵，需要存储项个数为 |V|(|V|+1)/2=Θ($|V|^2$) ，但采用这种方法并不能降低空间复杂度。

邻接矩阵同样可以表示权重图。

- 若（u，v）∈E，则a_{uv}=w(u,v)
  -表示将边（u，v）的权重w(u,v)存放在邻接矩阵第u行第v列。
- 若（u，v）∉E，则a_{uv}=NIL，通常用0或∞表示,视应用情况而定。

邻接矩阵表示法较邻接链表更简单，图规模较小倾向于使用邻接矩阵。无权图的邻接矩阵每一项只需要一个比特位。

`ps：第三版译本中此处有误，无权图写作了无向图。

- 若（u，v）∈E，a_{uv}=1
- 若（u，v）∉E，a_{uv}=0

表示属性

属性可以表示为“对象.属性”，如v.d（结点v的属性d），（u，v）.f（边（u，d）的属性f）。
若使用邻接链表，可以使用额外数组表示节点属性。如一个与Adj数组相对应的数组d[1..|V|]。如果与u相邻的结点都在Adj[u]中，则属性u.d存放在数组项的d[u]里。
在面向对象编程语言中，结点属性还可以表示为Vertex类下面一个子类中的实例变量。

20.2广度优先搜索

> 类似于树的层次遍历

给定一个图G=（V，E）和一个可识别的源节点s。
广度优先搜索系统性的探索图G中的边，发现可从s到达的所有节点，可计算从源节点s到每个可达节点的距离（最少边数），同时生成一棵以s为根并且包含所有可到达节点的广度优先搜索树。
树中从s到v的简单路径，对应图G中从s到v的最短路径（包含边数最少的路径）。
广度优先搜索既可以用于有向图，也可以用于无向图。

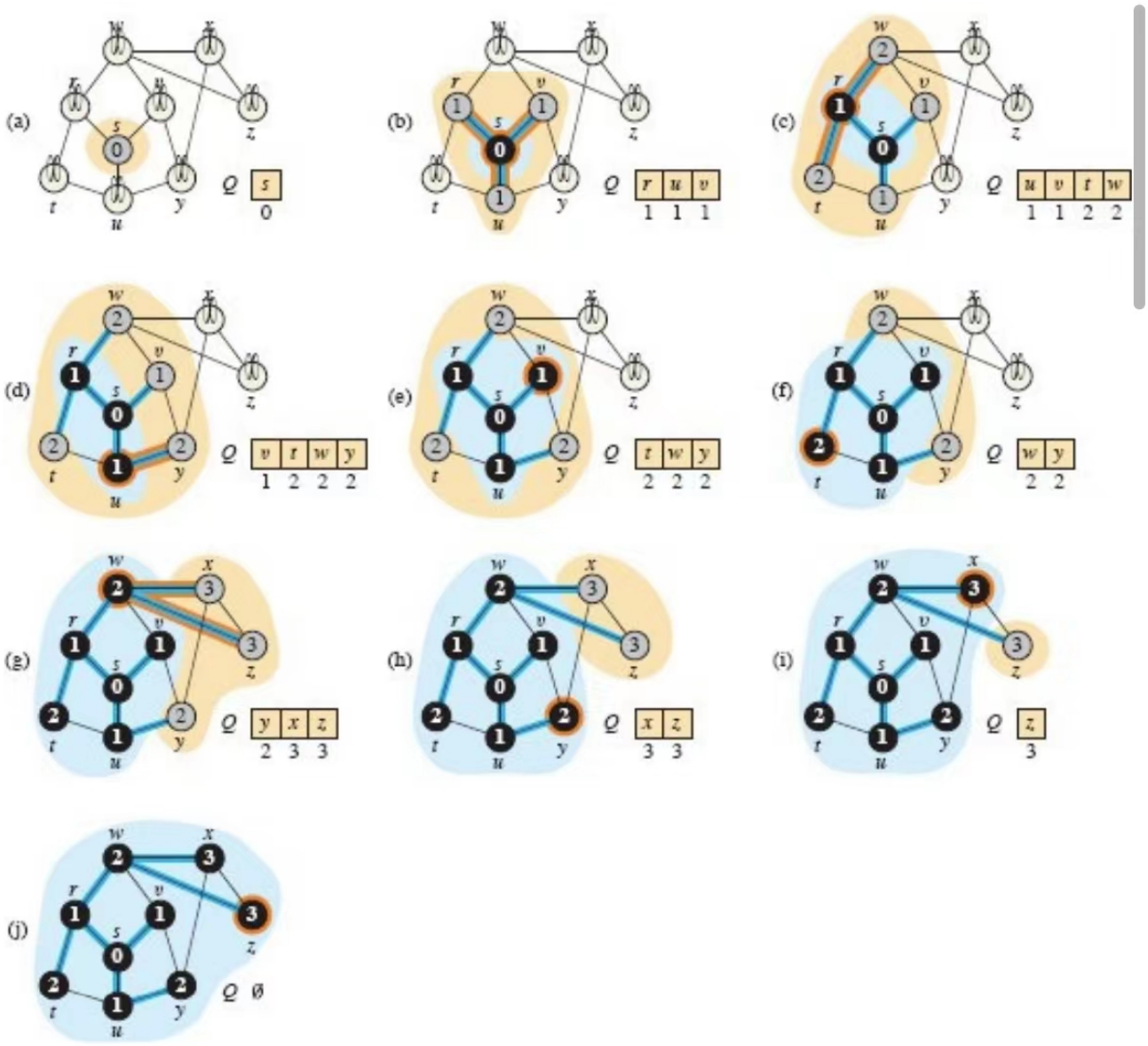
`ps：在求无权图单源最短路径问题中广度优先搜索比单源最短路径更加高效。

先发现距离s为k的所有节点，再发现距离s为k+1的所有节点（类比同心圆），直到发现所有从源节点可到达的点。

在概念上将每个结点涂上白色，灰色，黑色以跟踪算法进展。初始所有顶点为白色。当某顶点被发现则变为灰色。当某顶点领接表全部被检查则变为黑色。

执行广度优先搜索过程中构造广度优先搜索树。
开始树中仅有根节点及源节点s。扫描每发现一个白色节点v就将v及边（u，v）同时放入树中，称u为v的前驱/父节点，除了源节点外，每个从s可以到达的顶点有且只有一个父节点，若u为s到v的简单路径上的一个顶点，则u为v的祖先，v为u的后代。

BFS在无向图中的实现过程图示



BFS过程中需给每个节点u添加属性。

- u.color : u的颜色（白、灰、黑）
- u.d : s到u的距离
- u.π : u的前驱节点（u=s或u未发现，则u.π=NIL）

BFS伪代码

```
BFS(G, s)
    for each vertex u ∈ G.V - {s}
        u.color = WHITE
        u.d = ∞
        u.π = NIL
    s.color = GRAY
    s.d = 0
    s.π = NIL
```

```
    Q = ∅
    ENQUEUE(Q, s)
    while Q ≠ ∅
        u = DEQUEUE(Q)
        for each vertex v in G.Adj[u]    // search the neighbors of u
            if v.color == WHITE    // is v being discovered now?
                v.color = GRAY
                v.d = u.d + 1
                v.π = u
                ENQUEUE(Q, v)    // v is now on the frontier
        u.color = BLACK
```

广度优先搜索的结果取决于第12行顶点遍历的顺序，广度优先树可能不同，但每个节点的d确定。

BFS 过程可在队列 Q 变空之前的许多情况下终止。

- 每个顶点都有一个有限 d 值（每个顶点最多被发现一次，只有在被发现时才接收 d 值）。
- 队列 Q 为空
- 已经发现所有 |V| 个顶点。

`ps：提前终止不能改变 BFS 过程的时间复杂度

分析

- 对队列操作的时间为 O(|V|) （每个顶点入队一次，出队一次，入队和出队的时间为 O(1)，总共 |V| 个顶点）。
- 扫描邻接表的时间为 O(|E|) 。每个顶点出队时才对其邻接表进行扫描，所以每个邻接表只会被扫描一次，所有 |V| 个邻接表长度之和为 O(|E|)，
- 用于初始化操作的时间为 O(|V|) 。
  综上，BFS 过程的运行时间为 O(|V|+|E|)。

`ps：BFS中G为邻接链表表示，若为邻接矩阵则运行时间为O(|V|2)

最短路径

`ps：本节讨论的均为无权图，即所有边权重为1，距离为1.

定义从源点 s 到节点 v 的最短路径距离 δ(s,v) 为从 s 到 v 的所有路径中最少的边数。若从 s 到 v 没有路径，则 δ(s,v)=∞ 。称从 s 到 v 的长度为 δ(s,v) 的路径为s 到 v 的最短路径。

证明广度优先搜索可以正确计算出最短路径距离。

引理20.1

给定 G=(V,E) 为一个有向图或无向图，对于任一节点 s∈V ，任一边 (u,v)∈E ，有 δ(s,v)≤δ(s,u)+1 。
证明：类比三角形三边关系。当且仅当三点共线时取等。

引理20.2

给定 G=(V,E) 为一个有向图或无向图，若过程 BFS 以给定源点 s∈V 在 G 上运行，则在 BFS 运行过程中包括终止时，对于每一个节点 v∈V ， BFS 计算出的 v.d 的值始终满足 v.d≥δ(s,v) 。

`即证明节点在搜索树上的深度不小于最短路径。

证明（归纳法）

引理20.3

给定 G=(V,E) 为一个有向图或无向图，在 BFS 的执行过程中，队列 Q 包含顶点 ⟨v1,v2,...,vr⟩ ，其中 v1 在队头， vr 在队尾，有 vr.d≤v1.d+1 且对于 i=1,2,...,r−1 ，有 vi.d≤vi+1.d 。

`证明队列中最多包含两个不同的d值

证明（归纳法，讨论出队入队两种情况）

引理20.4

给定 G=(V,E) 为一个有向图或无向图，在 BFS 的执行过程中，顶点 vi 和 vj 都加入到队列 Q 中，并且 vi 在 vj 之前入队，则在 vj 入队时，有 vi.d≤vj.d 。

`证明节点加入队列过程中d值随时间单调增长。

证明：根据引理20.3，以及过程 BFS 对每一个从源点 s 可达顶点 v ， v.d 计算且仅计算一次，且 v.d 为有限的，可证。

引理20.5（广度优先搜索的正确性）

给定 G=(V,E) 为一个有向图或无向图，若过程 BFS 以给定源点 s∈V 在 G 上运行，在 BFS 的执行过程中，发现每一个从 s 可达的顶点 v∈V ，并在 BFS 的执行终止时，对于所有 v∈V ，有 v.d=δ(s,v) 。此外，对于任一从 s 可达的顶点 v≠s ，从 s 到 v 的其中一条最短路径为从 s 到 v.π 的最短路径加上边 (v.π,v) 。

`反向说明深度不大于最短路径。

证明（反证法）：

广度优先树

图20.3中的蓝色部分构成了在 BFS 的执行过程的广度优先树。该树对应 π 属性。
形式化地，对于图 G=(V,E) 和源点 s ，我们定义图 G 的前驱子图为 Gπ=(Vπ,Eπ) ，其中
(20.2) $V_\pi=\{ v \in V : v.\pi \neq NIL \} \cup \{ s \}$
(20.3) $E_\pi=\{ (v.\pi , v) : v\in V_\pi-\{ s \} \}$
若 Vπ 由从源点 s 可达的顶点组成，对于所有 v∈Vπ ， Gπ 包含从 s 到 v 的唯一简单路径，也是从 s 到 v 的一条最短路径，则前驱子图 Gπ 是一棵广度优先树。在该树中， $|E\pi|=|V\pi|-1$ （详见定理B.2）， Eπ 中的边被称为树边

```
定理B.2(证明略）：
```

定理 B.2（自由树性质） 令 $G=(V, E)$ 是一个无向图。下面的描述是等价的。

1. G 是自由树。
2. G 中任何两顶点由唯一简单路径相连。
3. G 是连通的，但是从图中移除任意一条边得到的图均不连通。
4. G 是连通的，且 $|E|=|V|-1$。
5. G 是无环的，且 $|E|=|V|-1$。
6. G 是无环的，但是如果向 E 中添加任何一条边，均会造成图包含一个环。

引理20.6

给定 G=(V,E) 为一个有向图或无向图，若过程 BFS 以给定源点 s∈V 在 G 上运行，则在 BFS 运行过程中通过计算 π 属性构造出的前驱子图 Gπ 为一棵广度优先树。

证明：分析过程 BFS 第16行，根据定理B.2和定理20.5可证。

假定过程 BFS 已经构造出一棵广度优先树
伪代码 PRINT-PATH 打印从 s 到 v 的一条最短路径上的所有顶点

```
PRINT-PATH(G, s, v)
    if v == s
        print s
    else if v.π == NIL
        print "no path from" s "to" v "exists"
    else PRINT-PATH(G, s, v.π)
        print v
```

运行时间与路径上的顶点个数呈线性关系。

20.3深度优先搜索

```
类似于树的先序遍历
```

深度优先搜索优先沿着从最近发现的顶点 v 发出的一条边进行探索。一旦从 v 发出的所有边都被探索后，则需要回溯到到达 v 的前一个顶点进行继续探索。不断执行上述过程，直到从源点可达的所有顶点全部被发现。
若还有顶点未被发现，选择未被发现的顶点中的任一顶点作为新的源点继续上述过程。

```
ps：理论上，广度优先搜索也可以从多个源点开始搜索，深度优先搜索也可以限定从一个源点开始。
典型使用中{本书）广度优先搜索通常用来寻找从给定源点出发的最短路径及其前驱子图，源点的数量为一个；深度优先搜索通常作为另一个算法里的一个子程序，可以有多个源点。
ps.ps：多源广度优先搜索可以用来寻找距离所有源点距离和最小的顶点。
ps.ps.ps：多源深度优先搜索本质就是由多个单源深度优先搜索组成并同步执行。
```

对已发现顶点 u 的邻接表进行扫描时，每发现一个顶点 v ，需要设定 v.π=u 。
广度优先搜索的前驱子图是一棵树，深度优先搜索的前驱子图可能包含若干棵树（多源搜索）。

定义图 G=(V,E) 前驱子图为 Gπ=(V,Eπ)，其中 Eπ={(v.π,v) : v∈V∧v.π≠NIL} 。
深度优先搜索前驱子图为一个有若干棵深度优先树的深度优先森林， Eπ 中的边被称为树边。

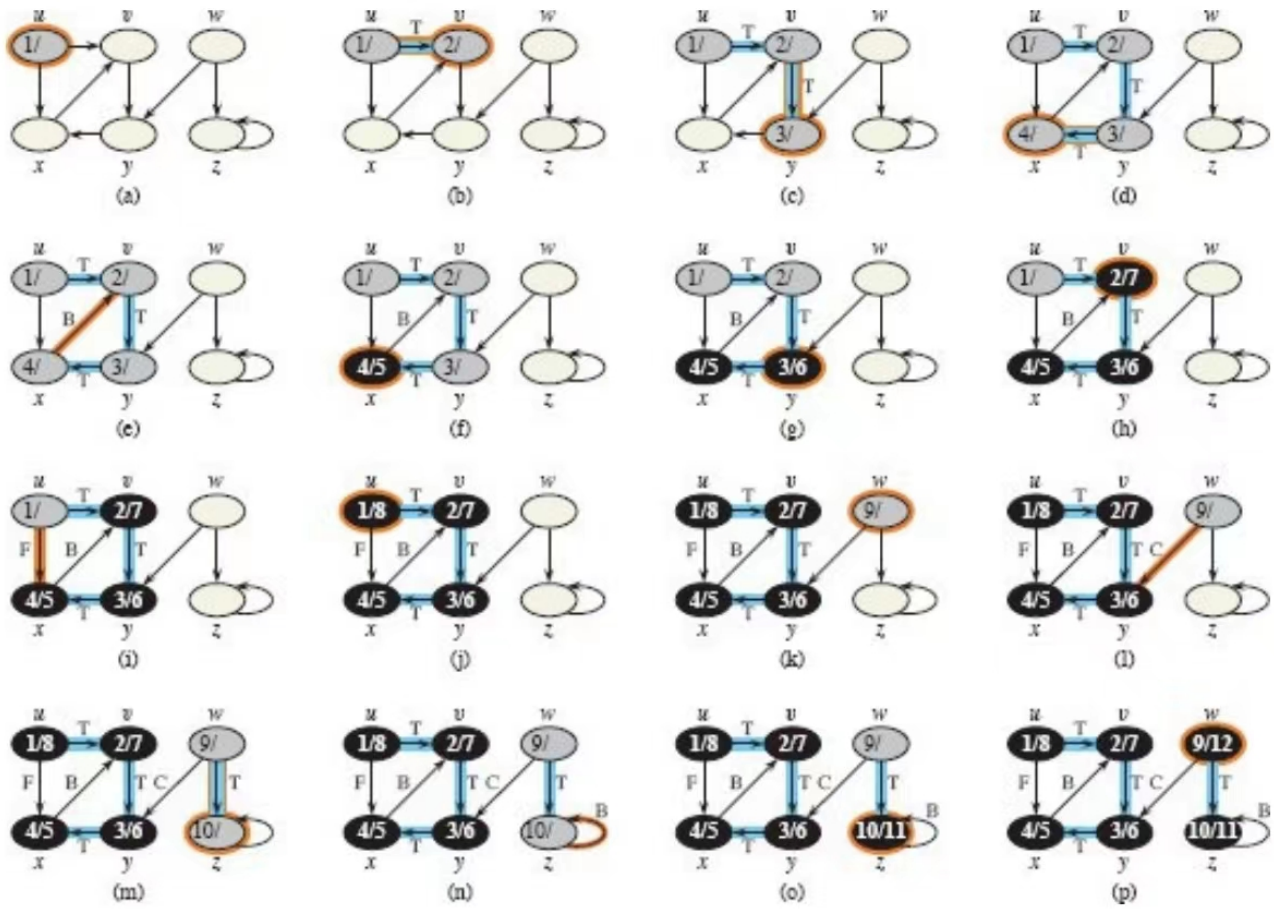
将所有顶点着色为白色、灰色和黑色。初始所有顶点为白色。当某顶点被发现则变为灰色。当某顶点领接表全部被检查则变为黑色。每个顶点只在一棵深度优先树中出现，所有深度优先树不相交。

每一个顶点 v 有两个时间戳：

- v.d 记录了 v 第一次被发现（变灰）的时间
- v.f 记录了 v 被搜索结束（变黑）的时间。

`ps：处理强连通问题的Targan算法也运用了时间戳。

对于 |V| 个顶点中每一个顶点 u，都只有一个发现事件和一个完成事件，时间戳范围为 1 到 2|V| 之间的整数，有u.d<u.f (20.4)。
时刻 u.d 之前， u 为白色
时刻 u.d 到 u.f 之间，u 为灰色
时刻 u.f 之后， u 为黑色



过程 DFS 的伪代码如下（全局变量 time 用来打时间戳）：

```
DFS(G)
    for each vertex u ∈ G.V
        u.color = WHITE
        u.π = NIL
```

```
    time = 0
    for each vertex u ∈ G.V
        if u.color == WHITE
            DFS-VISIT(G, u)

DFS-VISIT(G, u)
    time = time + 1    // white vertex u has just discovered
    u.d = time
    u.color = GRAY
    for each vertex v ∈ G.Adj[u]   // explore each edge (u, v)
        if v.color == WHITE
            v.π = u
            DFS-VISIT(G, v)
    time = time + 1
    u.f = time
    u.color = BLACK    // blacken u; it is finished
```

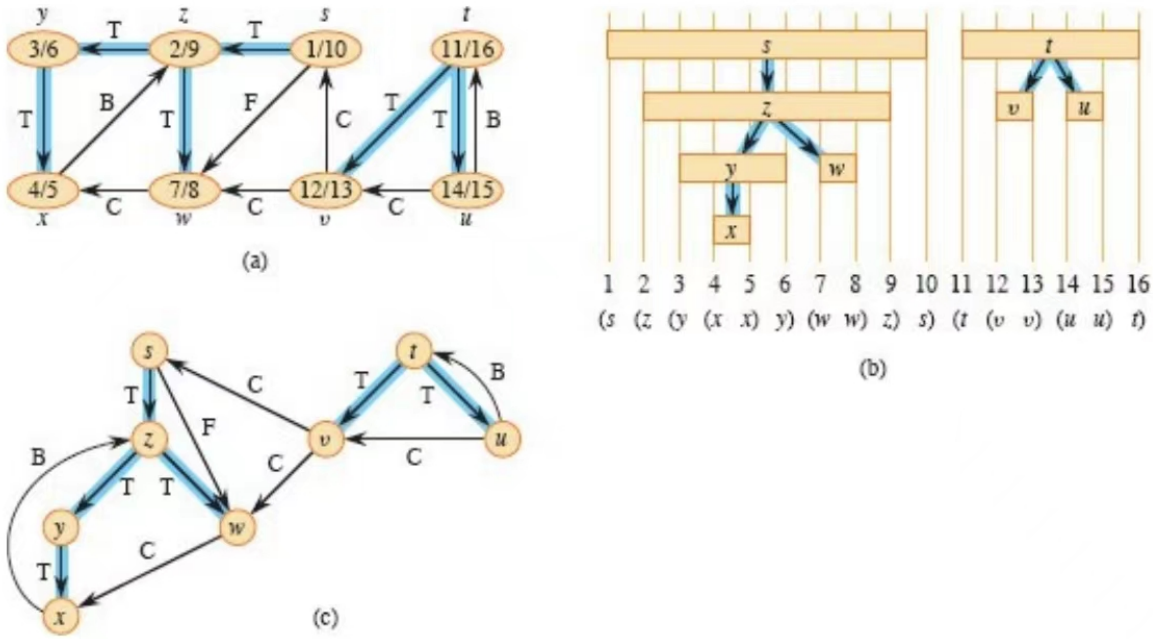
DFS 的运行时间为 Θ(|V|+|E|) 。

深度优先搜索的性质

- 深度优先搜索生成的前驱子图 Gπ 为一个有若干棵树的森林，因为深度优先树的结构与 DFS-VISIT 递归调用的结构完全对应， u=v.π 当且仅当 DFS-VISIT(G,v) 在搜索 u 的邻接表时被调用

- 在深度优先森林中，v 是 u 的后代，当且仅当 v 在 u 为灰色的时间段里被发现。

- 顶点的发现时间和完成时间具有括号化结构（20.5（b）），用左括号表示顶点被发现，右括号表示顶点搜索已完成，发现和完成中间的表达式适当地嵌套在这对括号中。



> 图20.5，
> a图为有向图深度优先搜索图示，图中注明了时间戳，边类型
> b图括号化结构，图中给出的边都是树边，对应较小区间的结点是较大区间的结点的后代
> c图给出深度优先树中所有的树边、往下的从祖先指向后代的前向边和往上的从后代指向祖先的后向边。

定理20.7（括号化定理）

深度优先搜索中，对于任意两个顶点 u 和 v ，以下三种情况只有一种成立：

- 若 [u.d,u.f]∩[v.d,v.f]=ø ，则在深度优先森林中， u 不是 v 的后代， v 也不是 u 的后代。
- 若 [u.d,u.f]⊂[v.d,v.f] ，则在深度优先森林中， u 是 v 的后代。
- 若 [v.d,v.f]⊂[u.d,u.f] ，则在深度优先森林中， v 是 u 的后代。

证明：

- u.d<v.d
  - 若 v.d<u.f ，即 v 在 u 为灰色的时间段里被发现，意味着 v 是 u 的后代，有 v.f<u.f ，即 [u.d,u.f]⊂[v.d,v.f] 。
  - 若 v.d>u.f ，即 v 在 u 为黑色后被发现，有 u.d<u.f<v.d<v.f ，即 [u.d,u.f]∩[v.d,v.f]=ø 。
- v.d<u.d
  - 若 u.d<v.f ，即 u 在 v 为灰色的时间段里被发现，意味着 u 是 v 的后代，有 u.f<v.f ，即 [v.d,v.f]⊂[u.d,u.f] 。
  - 若 u.d>v.f ，即 u 在 v 为黑色后被发现，有 v.d<v.f<u.d<u.f ，即 [v.d,v.f]∩[u.d,u.f]=ø 。

推论20.8（后代区间的嵌套）

在有向图或无向图 G=(V,E) 的深度优先森林中，顶点 v 是顶点 u 的真后代当且仅当 u.d<v.d<v.f<u.f 。

证明：
由定理20.7可得。

定理20.9（白色路径定理）

在有向图或无向图 G=(V,E) 的深度优先森林中，顶点 v 是顶点 u 的后代当且仅当在搜索发现 u 的时刻 u.d ，存在一条从 u 到 v 的全部由白色顶点构成的路径。

证明：

- 充分性：
  - 若 u=v ，即路径上只有一个顶点 u ，则在时刻 u.d ，有 u 为白色
  - 若 v 为 u 的任一真后代，有 u.d<v.d ，则根据推论20.8，有 v 为白色，从 u 到 v 的路径上的全部顶点为白色。
- 必要性：

- 若在时刻 u.d ，存在一条从 u 到 v 的全部由白色顶点构成的路径。先假设 v 在深度优先树中不是 u 的后代，设 v 为白色路径上第一个不是 u 的后代的顶点，设 v 的前驱为 w ， w 为 u 的后代（ w 可能为 u ），根据推论20.8，有 u.d<v.d<w.f≤u.f ，根据定理20.7，有 [v.d,v.f]⊂[u.d,u.f] ，根据推论20.8， v 为 u 的后代。

边的分类

对在图 G 上运行深度优先搜索生成的深度优先森林 Gπ ，定义四种边的类型：

1. 树边： Gπ 上的边，若顶点 v 是通过扫描边 (u,v) 第一次被发现，则 (u,v) 是一条树边。
2. 后向边：若边 (u,v) 为连接 u 和它在深度优先树中的一个祖先 v 的边，包括自循环（u=v ），则(u,v) 是一条后向边。（子→父）
3. 前向边：若边 (u,v) 为连接 u 和它的一个真后代 v 的非树边，则(u,v) 是一条后向边。（父→子）
4. 横向边：其它所有边，可连接同一深度优先树中的顶点，只要其中一个顶点不是另一个顶点的祖先，它们也可以连接不同深度优先树中的顶点。

见图20.5（c），中 T 表示树边， B 表示后向边， F 表示前向边， C 表示横向边。

第一次探索边 (u,v) 时，我们可以根据 v 的颜色判断边 (u,v) 的类型：

- 若 v 为白色，则 (u,v) 为树边。
- 若 v 为灰色，则 (u,v) 为后向边。
- 若 v 为黑色，则 (u,v) 为前向边或横向边。

情况一，根据算法推理可得。
情况二，灰色顶点总是形成一条线性的后代链，这些顶点保存在当前活跃的DFS-VISIT 的栈中。
情况三，详见练习20.3-5。

20.3-5：证明在一个有向图中，边 (u,v) 是：

a. 树边或前向边当且仅当 u.d < v.d < v.f < u.f 。
b. 后向边当且仅当v.d≤u.d < u.f≤v.f 。
c. 横向边当且仅当 v.d < v.f < u.d < u.f 。

定理20.10

在对无向图 G=(V,E) 的深度优先搜索中， G 中的每一条边要么是树边，要么是后向边

证明：
设 (u,v) 是一条边，若访问边 (u,v) 时 v 第一次被发现，则 (u,v) 为一条树边，若访问边 (u,v) 时 v 已经被发现，则 (u,v) 为一条后向边。

20.4拓扑排序

有向无环图 G=(V,E)所有顶点的一种线性排序，只要存在边(u,v)，就让u排在v前面。
仅限于有向无环图。若有向图中包含环，则无法进行进行线性排序。可以将拓扑排序视为将图中所有顶点在一条水平线上排开，有向边全部都是从左指向右。拓扑排序与之前讨论的排序是不同的。
ps：拓扑排序可能不唯一

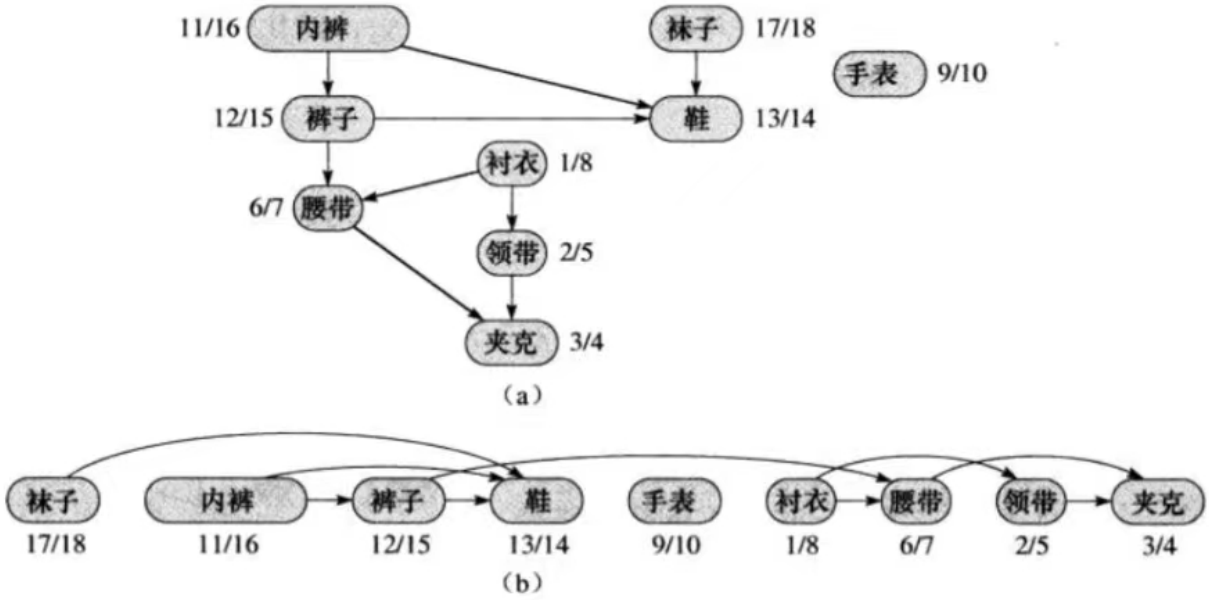


图 22-7 (a)Bumstead 教授对自己每天早上的穿衣进行的拓扑排序。每条有向边(u, v)表明服装 u 必须在服装 v 之前穿上。深度优先搜索的发现时间和完成时间注明在每个结点旁边。(b)以拓扑排序展示的同一个图，所有的结点按照其完成时间的逆序被排成从左至右的一条水平线。所有的有向边都从左指向右

深度优先搜索：循环以当前仍未搜索的节点为起点，进行DFS，然后逆序把节点存入列表中
DFS实现伪代码：

```
TOPOLOGICAL-SORT(G)
    call DFS(G) to compute finish times v.f for each vertex v
    as each vertex is finished, insert it onto the front of a linked list
    return the linked list of vertices
```

由于深度优先搜索运行时间为 Θ(|V|+|E|)，将每个顶点插入到链表最前端时间为 O(1)，总计有 |V| 个顶点，因此过程 TOPOLOGICAL-SORT 的运行时间为 Θ(|V|+|E|)+O(|V|)=Θ(|V|+|E|) 。

拓扑排序中的深度优先搜索是一个自底向上的过程，每个顶点插入到链表最前端

也可以用广度优先搜索实现自顶向下的过程，每个顶点插入到链表最后端

思考：如何使用广度优先搜索实现拓扑排序？时间复杂度？

证明拓扑排序正确性，证明以下有向无环图相关引理。

引理20.11

有向图 G 是无环的当且仅当对 G 进行深度优先搜索不产生后向边。
证明：

- 充分性：假设深度优先搜索产生了一条后向边 (u,v)，则在深度优先树中顶点 v 是顶点 u 的祖先，则 G 中存在一条从 v 到 u 的路径，该路径和边 (u,v) 一起构成了一个环，矛盾
- 必要性：假设 G 包含一个环路 c，设 v 是环路上第一个被发现的顶点，设 (u,v) 是环路中指向 v 的一条边，在时刻 v.d，环路 c 中顶点形成一条从 v 到 u 的一条全白色顶点的路径，根据白色路径定理， u 将在深度优先树中成为 v 的后代，因此 (u,v) 为一条后向边。

定理20.12

拓扑排序算法生成的是有向无环图的拓扑排序
证明：
假设有向无环图 G=(V,E) 运行 DFS 来计算顶点的完成时间，只需要证明对于任意两个顶点 u,v∈V ，若 G 中存在一条有向边 (u,v) ，则 v.f<u.f 。当 DFS 探索任意一条边 (u,v) 时， v 不可能为灰色，否则 v 是 u 的祖先，即 (u,v) 为后向边，与引理20.11矛盾。因此 v 只能是白色或者黑色。若 v 为白色，则 v 是 u 的后代，有 v.f<u.f 。若 v 为黑色，则 v 已被探索完成，而 u 还在探索过程中，有 v.f<u.f 。综上，对于 G 中任意一条有向边 (u,v) ，有 v.f<u.f 。得证

20.5强连通分量

深度优先搜索树的经典应用：将有向图分解为强连通分量。
有向图 G=(V,E) 的一个强连通分量(strongly connected component, SCC)是一个最大顶点集 C⊆V ，对于每一对顶点 u,v∈C ， u⇝v 且 v⇝u ，即 u 和 v 相互可达。
给定图 G=(V,E) 的一个邻接表表示，创造 G 的转置 $G^T=(V,E^T)$ 的时间为 Θ(|V|+|E|) 。 G 和 G^T 有完全相同的强连通分量。

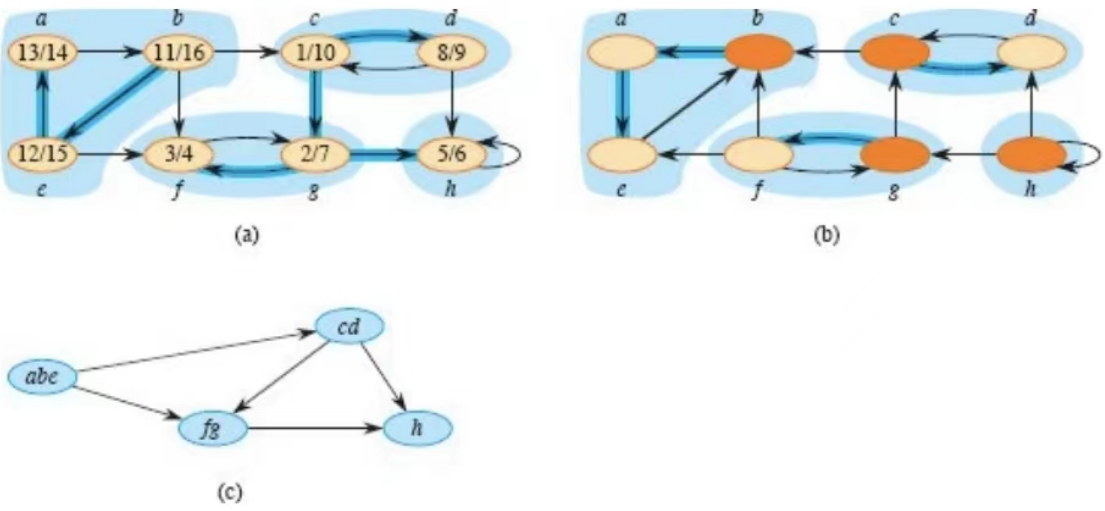


Figure 20.9 (a) A directed graph G. Each region shaded light blue is a strongly connected component of G. Each vertex is labeled with its discovery and finish times in a depth-first search, and tree edges are dark blue. **(b)** The graph G^T, the transpose of G, with the depth-first forest computed in line 3 of STRONGLY-CONNECTED-COMPONENTS shown and tree edges shaded dark blue. Each strongly connected component corresponds to one depth-first tree. Orange vertices b, c, g, and h are the roots of the depth-first trees produced by the depth-first search of G^T. **(c)** The acyclic component graph G^{SCC} obtained by contracting all edges within each strongly connected component of G so that only a single vertex remains in each component.

分量图(component graph) $G^{SCC}=(V^{SCC},E^{SCC})$ 定义如下：

假设 G 有强连通分量 $C_1,C_2,\ldots,C_k$ 。

- 顶点集 $V^{SCC}=\{v_1,v_2,\ldots,v_k\}$ ，满足顶点 $v_i\in C_i$ 。
- 边集 $E^{SCC}=\{(v_i,v_j)\mid x\in C_i, y\in C_j, (x,y)\in E, v_i\in C_i, v_j\in C_j, v_i,v_j\in V^{SCC}, i\neq j\}$ 。

ps：分量图为简化后的有向图，把每个强连通分量压缩成一个顶点，以便研究不同强连通分量之间的关系。代表每一个强连通分量的顶点理论上可以是该强连通分量中任意一个顶点

计算强连通分量过程伪代码：

```
STRONGLY-CONNECTED-COMPONENTS(G)
1 call DFS(G) to compute finish times u.f for each vertex u
2 create G^T
3 call DFS(G^T), but in the main loop of DFS, consider the vertices in
    order of decreasing u.f (as computed in line 1)
4 output the vertices of each tree in the depth-first forest formed in
    line 3 as a separate strongly connected component
```

分量图无环，见下：

引理20.13

令 C 和 C′ 为有向图 G=(V,E) 的两个强连通分量。令 u,v ∈C ， 令 u′,v′∈C′ ， 若 G 包含一条路径 u⇝u′ ， 则 G 不可能包含一条路径 v′⇝v 。

证明：
反证法。若 G 包含一条路径 v′⇝v ， 则它包含路径 u⇝u′⇝v′ 且 v′⇝v⇝u ， 即 u 和 v′ 相互可达，与 u 和 v′ 分属两个强连通分量矛盾。得证。

对于顶点集合 U⊆V ， 令 d(U)=min{ u.d : u∈U } 为 U 的发现时间， f(U)=max{ u.f : u∈U } 为 U 的结束时间。考虑 STRONLY-CONNECTED-COMPONENTS 的第1行的 DFS 。

引理20.14

令 C 和 C′ 为有向图 G=(V,E) 的两个强连通分量，若存在 (u,v)∈E ， 其中 u∈C′ 且 v∈C ， 则 f(C′)>f(C) 。

证明：分两种情况讨论。

- 若 d (C′)<d (C) ， 设 x 为 C′ 中最早被发现的顶点，在时刻 x.d ， C 和 C′ 中所有顶点都是白色的， C′ 中包含从 x 到 C′ 中每一个顶点的白色路径。因为 (u,v)∈E ， 对于任一顶点 w∈C ，有一条从 x 到 w 的仅包含白色顶点的路径 x⇝u→v⇝w 。根据白色路径定理， C 和 C′ 中所有顶点在深度优先树中都是 x 的后代，根据推论20.8， x.f=f (C′)>f (C) 。
- 若 d (C′)>d (C) ， 设 y 为 C 中最早被发现的顶点，在时刻 y.d ， C 中所有顶点都是白色的，C 中包含从 y 到 C 中每一个顶点的白色路径。根据白色路径定理， C 中所有顶点在深度优先树中都是 y 的后代，根据推论20.8， y.f = f (C) 。由于 d (C′)>d (C)=y.d ， 在时刻 y.d ， C′ 中所有顶点都是白色的。由于存在一条边 (u,v) 从 C′ 到 C ， 根据引理20.13，不存在一条从 C 到 C′ 的路径，即对于 C 中任一顶点都不存在一条到 y 的路径。因此在时刻 y.f ， C′ 中所有顶点都仍然是白色的。因此，对于任一顶点 w∈C′ ， 有 w.f > y.f ， 即 f (C′)>f (C) 。

综上，得证。

推论20.15

令 C 和 C′ 为有向图 G=(V,E) 的两个强连通分量，若 f(C)>f(C′)，则 ET 中不可能包含一条边 (v,u)，其中 u∈C′ 且 v∈C 。

证明：
引理20.14的逆否命题为令 C 和 C′ 为有向图 G=(V,E) 的两个强连通分量，若 f (C)>f (C′)，则 E 中不可能包含一条边 (u,v)，其中 u∈C′ 且 v∈C 。 $(u,v) \in E \Leftrightarrow (u,v) \in E^T$ ，得证。

定理20.16

过程 STRONLY-CONNECTED-COMPONENTS 能够正确计算有向图 G 的强连通分量。

证明：

- 过程第3行对 G^T 进行深度优先搜索时发现的深度优先树的数量进行归纳，其中每一棵深度优先树的顶点构成一个强连通分量。归纳假设为第3行生成的前 k 棵树都是强连通分量，当 k=0 时，显然成立。

- 归纳步骤：假设第3行生成的前 k 棵树都是强连通分量，考虑第3行生成的第 k+1 棵树，设其根结点为 u，所 u 在的强连通分量为 C，有 u.f=f (C)>f (C′)，其中 C′ 任一除 C 外尚未被访问的强连通分量。根据归纳假设，在时刻 u.d， C 中所有顶点都是白色的，根据白色路径定理，C 中所有顶点在深度优先树中都是 u 的后代，根据引理20.15，任一 G^T 中离开 C 的边只能指向已经访问过的强连通分量。因此，除 C 外的强连通分量中的顶点在对 G^T 的深度优先搜索过程中都不可能成为 u 的后代，因此， G^T 的以 u 为根深度优先树中的所有顶点构成一个强连通分量。得证。

- 换个角度考虑 STRONLY-CONNECTED-COMPONENTS 的第3行的 DFS，访问 $(G^T)^{SCC}$ 顶点的顺序为逆拓扑排序，由于 $((G^T)^{SCC})^T=G^{SCC}$ ，等价于访问 G^{SCC} 顶点的顺序为拓扑排序。得证