图

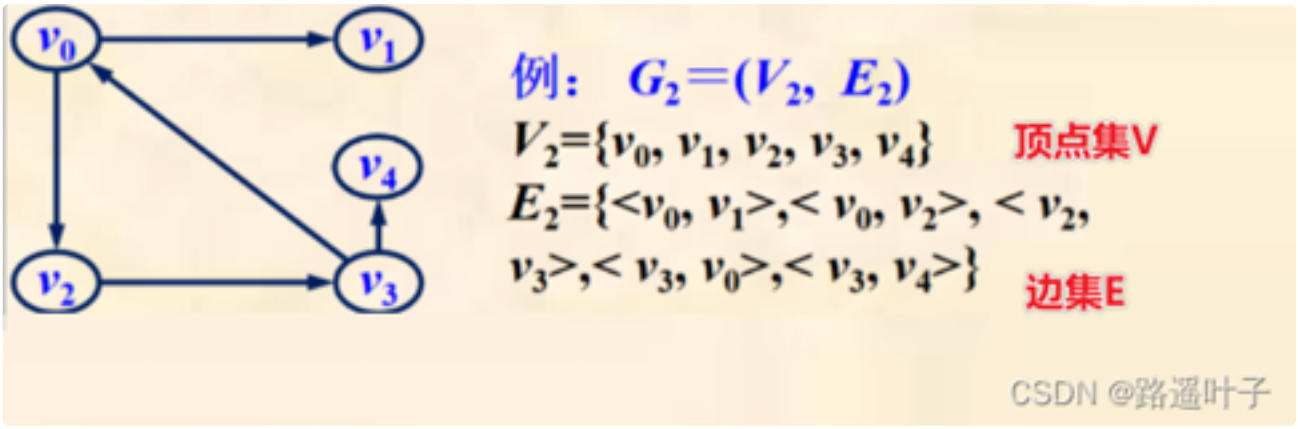
基本概念

- 图=顶点＋边
  $G=(V,E)$
  - V(G)->顶点集，|V|->顶点数(也称图的阶)
  - E(G)->边集，|E|->边数
- 线性表有空表，树有空树，但图没有空图
- 图的顶点集V一定非空，但边集E可以为空

分类

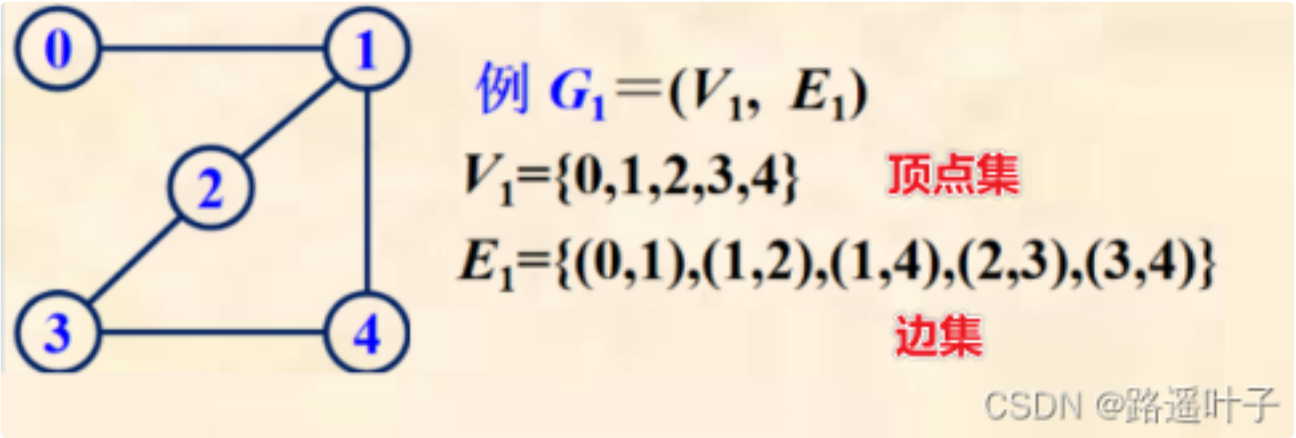
有向图

全部由有向边构成的图



无向图

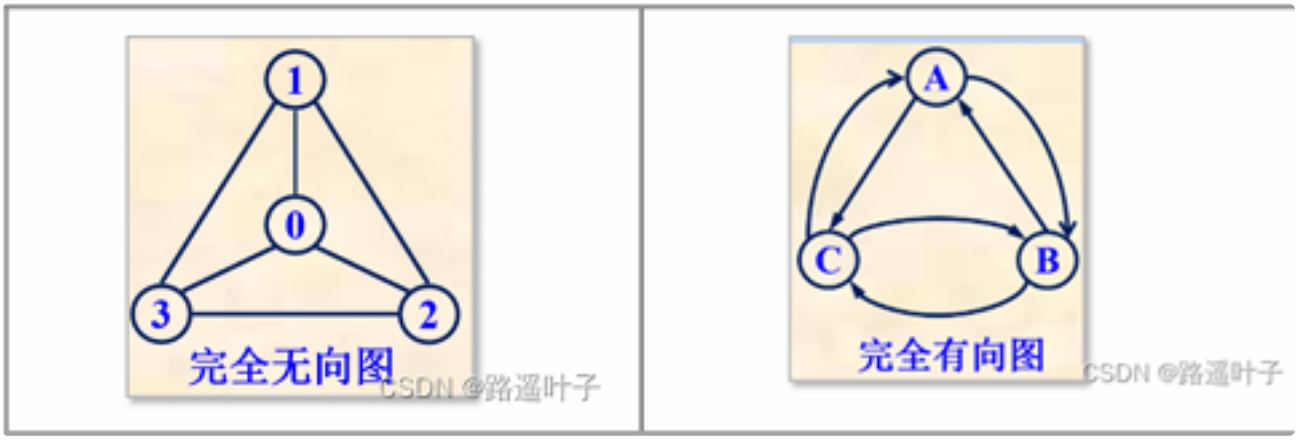
全部由无向边构成的图



简单图，多重图

简单图：不存在重复边，不存在顶点到自身的边
多重图：两个顶点之间的边数大于1

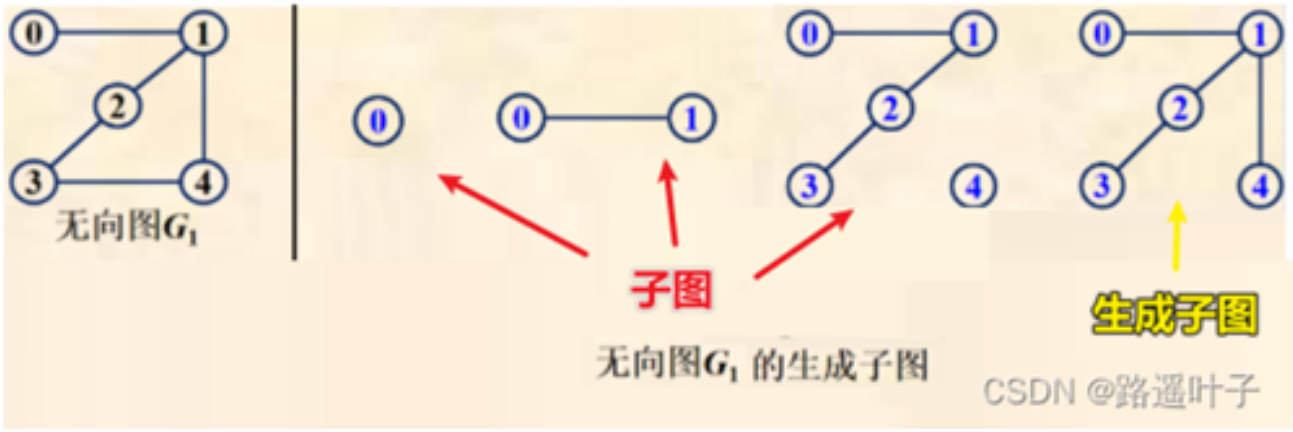
完全图(简单完全图)



完全无向图：边数 $\frac{n(n-1)}{2}$
完全有向图：边数 $n(n-1)$

子图和生成子图

G的子图：所有的顶点和边都属于图G的图
G的生成子图：含有G的所有顶点的子图



连通，连通图，连通分量(无向图)

v和w连通：无向图中,v到w的路径存在
连通图：图中任意两个顶点都是连通的
连通分量：无向图中的极大连通子图

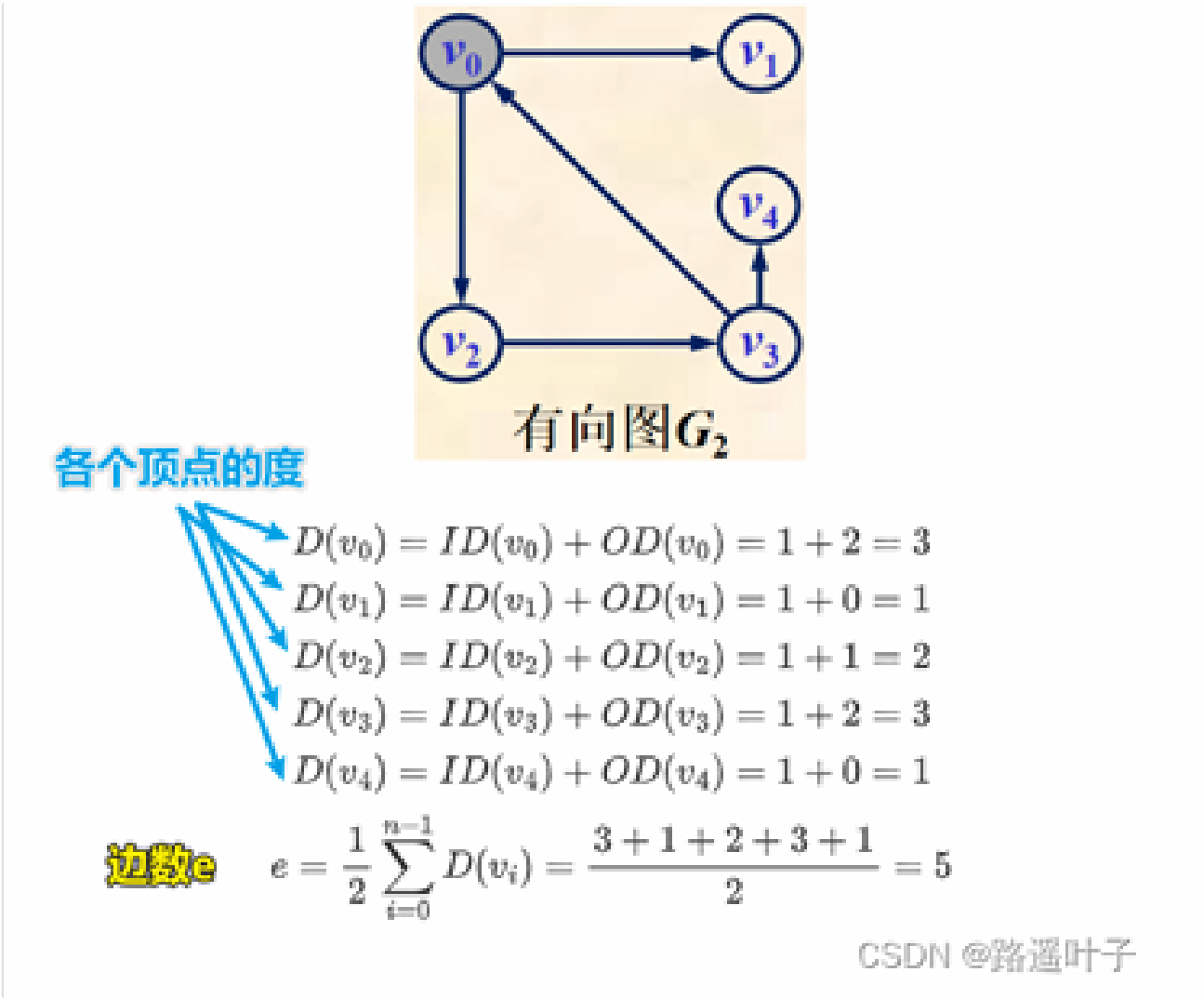
强连通，强连通图，强连通分量

v和w强连通：有向图中,从v到w和从w到v都有路径
强连通图：图中任何一对顶点都是强联通的
强连通分量：有向图中的极大连通子图

生成树和生成森林

生成树：包含图中全部顶点的一个极小连通图
生成森林：非连通图中,连通分量的生成树构成了非连通图的生成森林

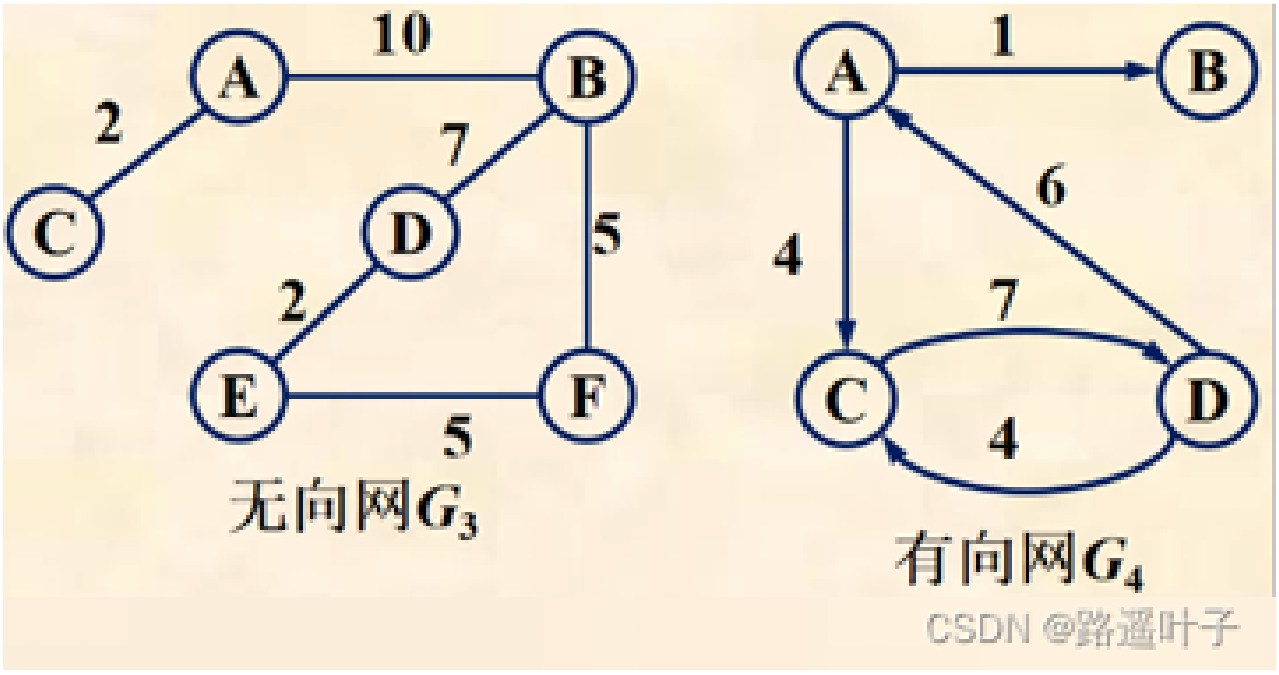
顶点的度，出度，入度



顶点的度：图中与该顶点相关联边的数目
入度：指向该顶点的边的数目
出度：从该顶点出去的边的数目
顶点的度= 出度 + 入度
对于具有n个顶点，e条边的有向图，出度和=入度和=e
对于具有n个顶点，e条边的无向图，度和=2e

边的权和网

权Weight：边上的数值
网Network：边上标识权的图



稠密图，稀疏图

稠密图：边多 稀疏图：边少

路径，路径长度和回路

路径Path：在一个图中，路径是从顶点u到顶点v所经过的顶点序列
路径长度：该路径上边的数目
回路：第一个顶点和最后一个顶点相同的路径

简单路径和简单回路

简单路径：顶点不重复出现的路径
简单回路：除第一个和最后一个顶点，其余顶点不重复出现的回路

距离

从u到v的距离：从u到v的最短路径长度

有向树

有向树：一个顶点的入度为0，其余顶点的入度均为1的有向图

Abstract

无向图边数*2 = 各顶点度数之和
有向图边数 = 各顶点的入度之和 = 各顶点的出度之和

一个连通图的生成树是一个极小连通子图，是无环的
对于一个有n个顶点的图 若是连通无向图，其边的个数至少为n-1（边最少即构成一棵树的情形）
若是强连通有向图，其边的个数至少是n（边最少即构成一个有向环的情形）

易错题

01. 图中有关路径的定义是（ ）。

A. 由顶点和相邻顶点序偶构成的边所形成的序列

B. 由不同顶点所形成的序列

C. 由不同边所形成的序列

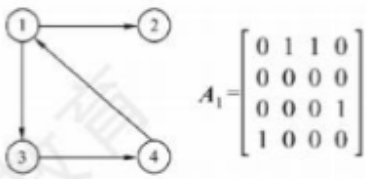
D. 上述定义都不是

14. 若具有 n 个顶点的图是一个环，则它有（ ）棵生成树。

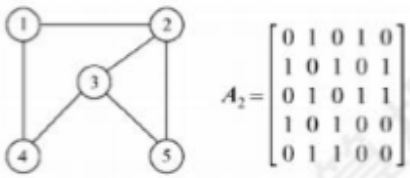
A. n^2　　B. n　　C. $n-1$　　D. 1

图的存储及基本操作

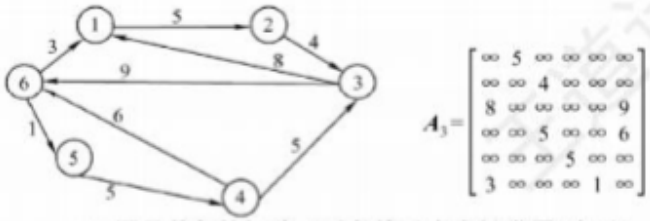
邻接矩阵



(a) 有向图G_1及其邻接矩阵



(b) 无向图G_2及其邻接矩阵



(c) 网及其邻接矩阵（对角线元素也经常用0表示）

图 6.5　有向图、无向图及网的邻接矩阵

邻接矩阵的遍历及顶点度的计算

```
#define MaxVertexNum 100
typedef char VertexType;
typedef int EdgeType;
typedef struct{
```

```
    VertexType vex[MaxVertexnum];
    EdgeType edge[MaxVertexNum][MaxVertexNum];
    int vexnum,arcnum;
}MGraph;
```

> **Info**
>
> 1. 在简单应用中，可直接用二维数组作为图的邻接矩阵（顶，点信息等均可省略）
> 2. 当邻接矩阵的元素仅表示相应边是否存在时，EdgeType可用值为0和1的枚举类型
> 3. 无向图的邻接矩阵是对称矩阵，对规模特大的邻接矩阵可采用压缩存储。
> 4. 邻接矩阵表示法的空间复杂度为 $O(n^2)$,其中n为图的顶点数。

$A^n[i][j]$ 等于顶点i到j的长度为n的路径的数目

邻接表

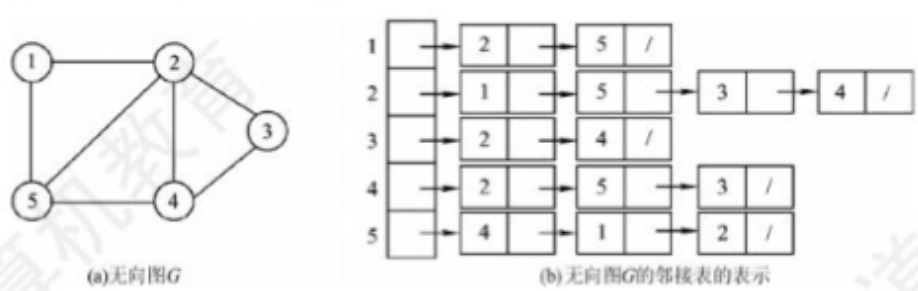


图 6.7　无向图邻接表表示法实例

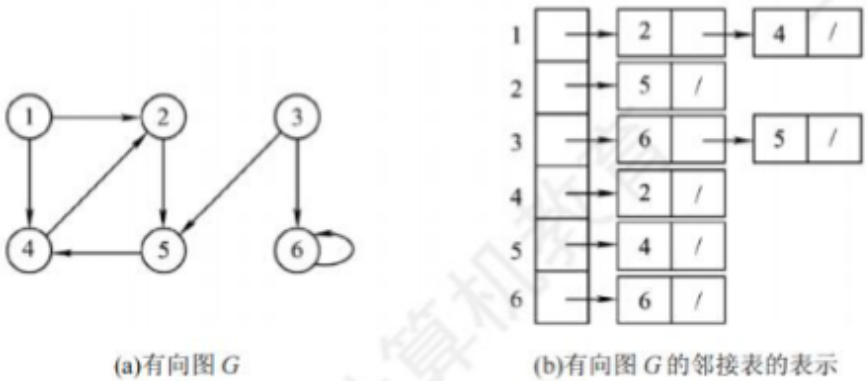


图 6.8　有向图邻接表表示法实例

实现

```
#define MaxVertexNum 100
typedef struct ArchNide{ //边表节点
    int adjvex; //指向节点的位置
```

```
        struct ArchNide *nextarc; //指向下一条弧的指针
}ArcNode;
typedef struct VNode{ //顶点表节点
        VertexType data; //顶点信息
        ArcNode *firstarc;//指向第一条依附该节点的指针
}VNode, AdjList[MaxVertexNum];
typedef struct {
        AdjList vertices;
        int vexnum,arcnum;
}ALGraph;
```

无向图的存储空间为 $O(|V|+2|E|)$,有向图 $O(|V|+|E|)$

十字链表法

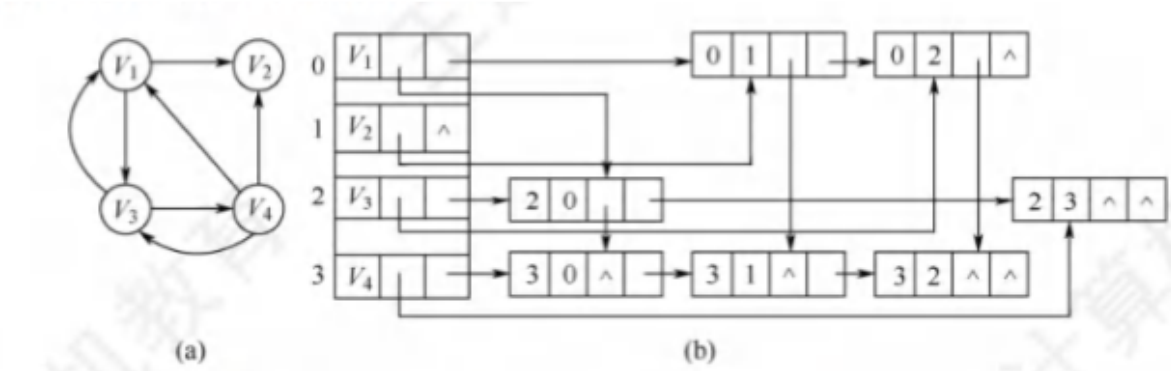


图 6.9　有向图的十字链表表示（弧结点省略 info 域）

邻接多重表

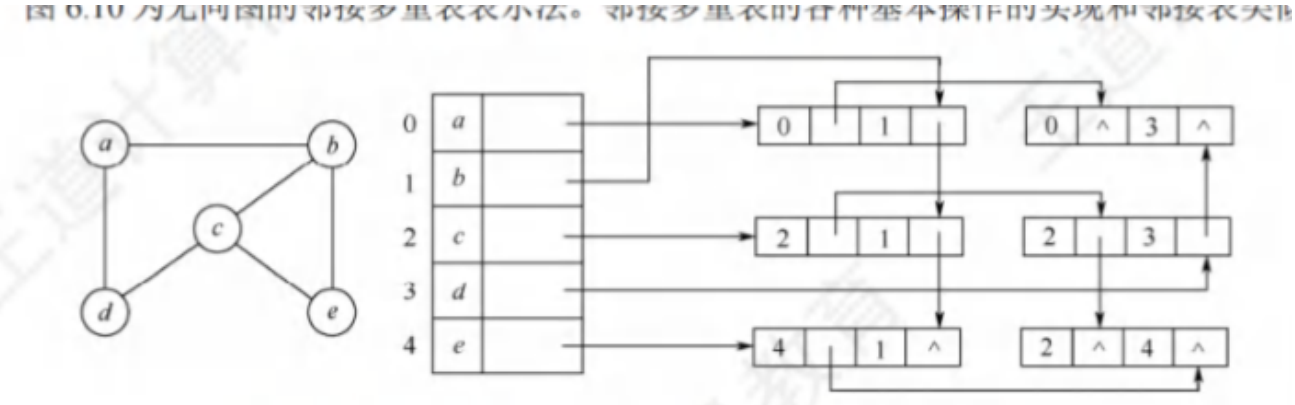


图 6.10　无向图的邻接多重表表示（边结点省略 info 域）

比较

表 6.1　图的四种存储方式的总结

	邻接矩阵	邻接表	十字链表	邻接多重表
空间复杂度	$O(\|V\|^2)$	无向图：$O(\|V\|+2\|E\|)$ 有向图：$O(\|V\|+\|E\|)$	$O(\|V\|+\|E\|)$	$O(\|V\|+\|E\|)$
找相邻边	遍历对应行或列的时间复杂度为 $O(\|V\|)$	找有向图的入度必须遍历整个邻接表	很方便	很方便
删除边或顶点	删除边很方便，删除顶点需要大量移动数据	无向图中删除边或顶点都不方便	很方便	很方便
适用于	稠密图	稀疏图和其他	只能存有向图	只能存无向图
表示方式	唯一	不唯一	不唯一	不唯一

图的基本操作

Adjacent(G,x,y)	- 判断是否存在边
Neighbors(G,x)	- 列出图中与结点x邻接的边
InsertVertex(G,x)	- 在图中插入顶点x
DeleteVertex(G,x)	- 在图中删除顶点x
AddEdge(G,x,y)	- 若无向边或有向边不存在，则向图中添加该边
RemoveEdge(G,x,y)	- 若无向边或有向边存在，则向图中删除该边
NextNeighbor(G,x,y)	- 假设图中顶点y是顶点x的一个邻接点 - 返回除y外的顶点x的下一个邻接点的顶点号 - 若y是x的最后一个邻接点，则返回1
Get_Edge_Value(G,x,y)	- 获取图中边对应的权值
Set_Edge_Value(G,x,y,v)	- 设置图中边对应的权值

代码题

【2021 统考真题】已知无向连通图 G 由顶点集 V 和边集 E 组成，$|E|>0$，当 G 中度为奇数的顶点个数为不大于 2 的偶数时，G 存在包含所有边且长度为 $|E|$ 的路径（称为 EL 路径）。设图 G 采用邻接矩阵存储，类型定义如下：

```
typedef struct{                          //图的定义
    int  numVertices,numEdges;           //图中实际的顶点数和边数
    char VerticesList[MAXV];             //顶点表。MAXV 为已定义常量
    int Edge[MAXV][MAXV];                //邻接矩阵
  }MGraph;
```

请设计算法 int IsExistEL(MGraph G)，判断 G 是否存在 EL 路径，若存在，则返回 1，否则返回 0。要求：

1）给出算法的基本设计思想。

2）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

3）说明你所设计算法的时间复杂度和空间复杂度。

思路

遍历

代码

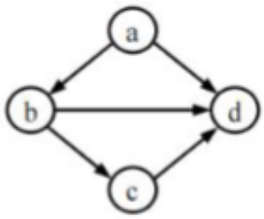
```
int IsExistEL(MGraph G)
{
        int degree,i,j,count;
        for(i=0;i<G.numVertices;i++)
        {
                degree =0;
                for(j=0;j<G.numVertices;j++)
                        degree +=G.Edge[i][j];
                if(degree&2!=0)
                        count++;
        }
        if(count==0||count==2)
                return 1;
        else
```

```
        return 0;
}
```

【2023 统考真题】已知有向图 G 采用邻接矩阵存储，类型定义如下：

```
typedef struct{                          //图的类型定义
    int numVertices,numEdges;            //图的顶点数和有向边数
    char VerticesList[MAXV];             //顶点表，MAXV 为已定义常量
    int Edge[MAXV][MAXV];                //邻接矩阵
}MGraph;
```

将图中出度大于入度的顶点称为 K 顶点。例如，在下图中，顶点 a 和 b 为 K 顶点。



请设计算法 int printVertices(MGraph G)，对给定的任意非空有向图 G，输出 G 中所有的 K 顶点，并返回 K 顶点的个数。要求：

1）给出算法的基本设计思想。

2）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

思路

行列遍历

代码

```
int printVertices(MGraph G)
{
    int degree,outdegree,k,m,count=0;
    for(k=0;k<G.numVertices;k++)
    {
        indegree=outgree=0;
        for(m=0;m<G.numVertices;m++)
            outdegree+=G.Edge[k][m];
        for(m=0;m<G.numVertives;m++)
            indegree+=G.Edge[m][k];
        if(outdegree>indegree)
            printf("%c",G.VerticesList[k]);
            count++;
    }
    return count;
}
```

图的遍历

广度优先搜索(BFS)

相当于树的层次遍历

遍历过程

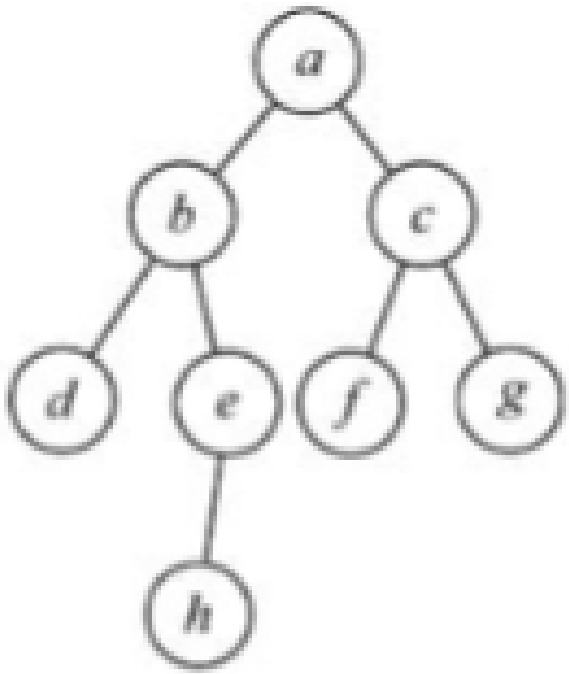


图 6.11 一个无向图 G

a->b->c->d->e->f->g

性能分析

邻接表：$O(V+E)$ 临接矩阵 $O(V^2)$

- BFS可以求解非带权图的单源最短路径问题

广度优先生成树

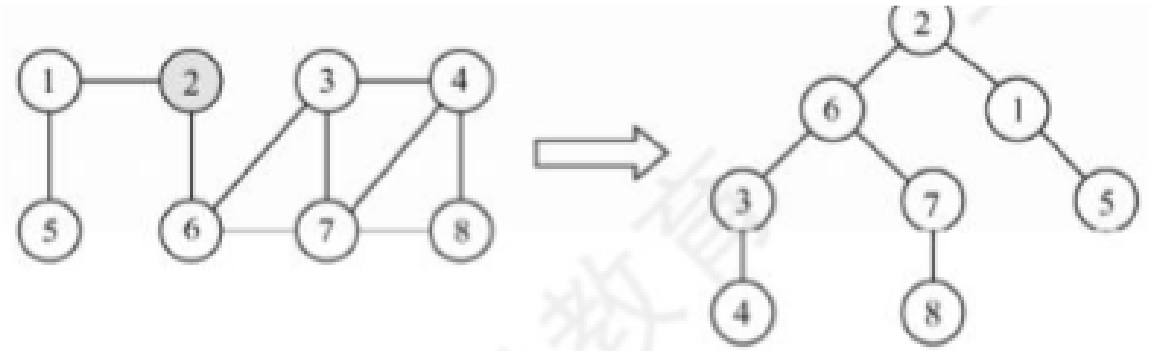


图 6.12　图的广度优先生成树

邻接矩阵的生成树唯一，邻接表不唯一

深度优先搜索(DFS)

类似于树的先序遍历

深度优先遍历的过程

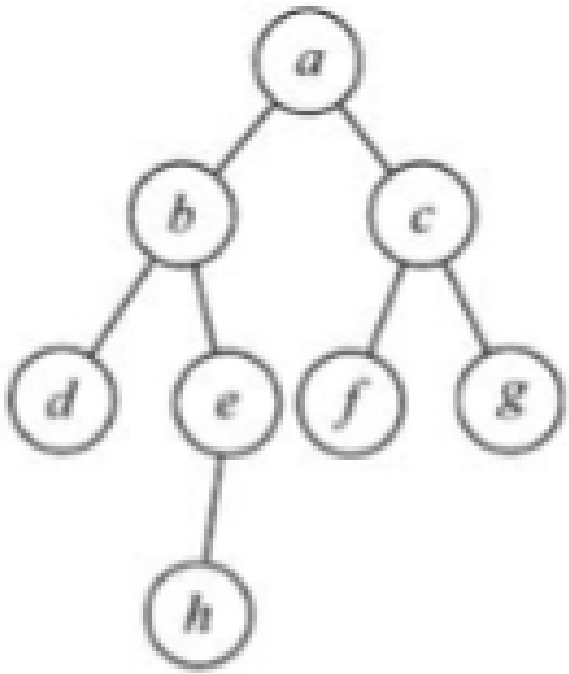


图 6.11　一个无向图 G

a->b->d->e->h->c->f->g

DFS算法性能分析

需要递归工作栈，空间复杂度 $O(V)$
邻接表：$O(V+E)$ 临接矩阵 $O(V^2)$

深度优先的生成树和生成森林

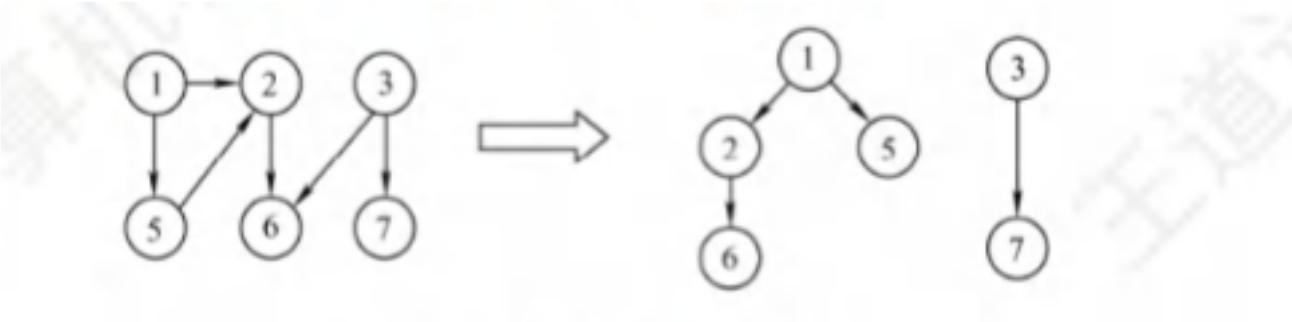


图 6.13　图的深度优先生成森林

图的应用

AOE网和AOV网

AOV不带权值，AOE带权值（都是有向无环图）

- 若AOE网只有一个入度为零的点，则称其为源点，同理出度为零则称汇点

最小生成树(MST)

Abstract

- 存在权值相同边MST不唯一，否则唯一
- 权值之和唯一
- 边数等于顶点数减一

Prim算法

以顶点为中心扩展，选最小加入

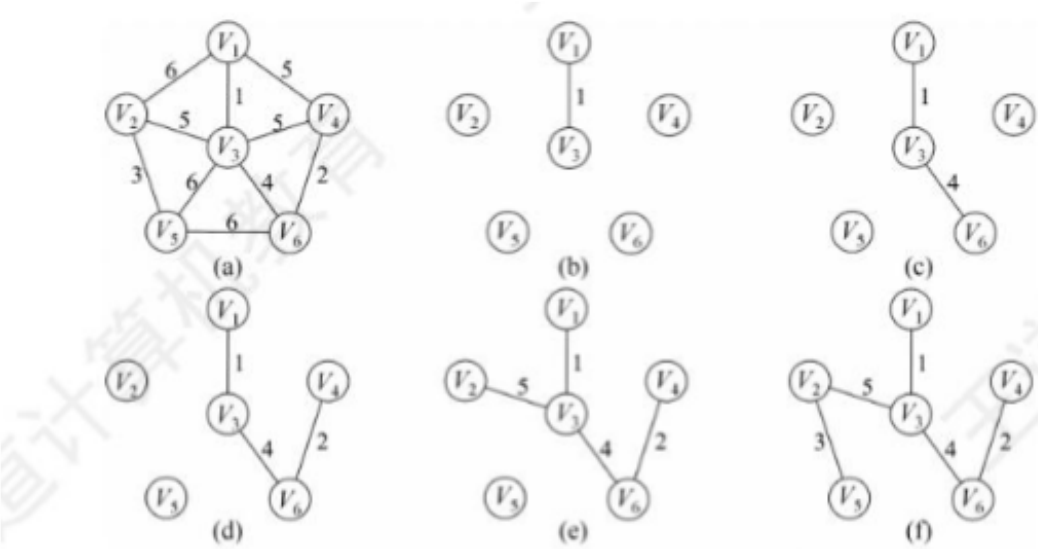


图 6.15　Prim 算法构造最小生成树的过程

时间复杂度 $O(V^2)$（适用于边稠密）

Kruskal算法

权值递增次序选择合适边

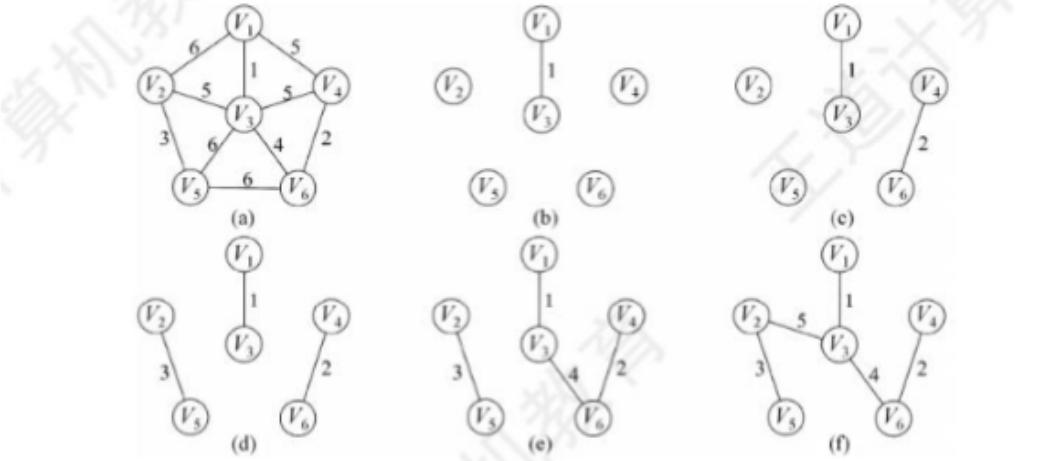


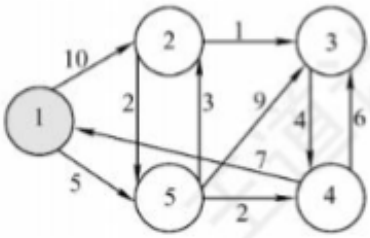
图 6.16　Kruskal 算法构造最小生成树的过程

时间复杂度 $O(Elog_2^E)$（适用于边稀疏）

最短路径

无向图直接用BFS搜索

Dikstra算法



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

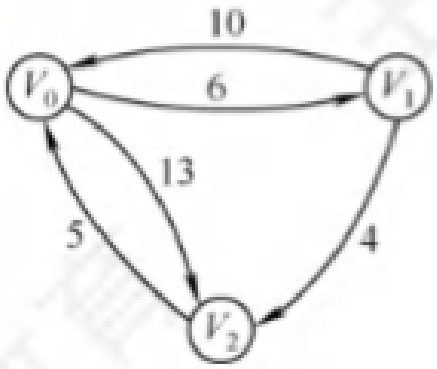
图 6.17　应用 Dijkstra 算法图

表 6.2　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}

时间复杂度 $O(V^2)$

Floyd算法



(a) 有向图 G

$$\begin{bmatrix} 0 & 6 & 13 \\ 10 & 0 & 4 \\ 5 & \infty & 0 \end{bmatrix}$$

(b) G 的邻接矩阵

图 6.19　带权有向图 G 及其邻接矩阵

表 6.3　Floyd 算法的执行过程

A	$A^{(-1)}$			$A^{(0)}$			$A^{(1)}$			$A^{(2)}$		
	V_0	V_1	V_2	V_0	V_1	V_2	V_0	V_1	V_2	V_0	V_1	V_2
V_0	0	6	13	0	6	13	0	6	**10**	0	6	10
V_1	10	0	4	10	0	4	10	0	4	**9**	0	4
V_2	5	∞	0	5	**11**	0	5	11	0	5	11	0

时间复杂度 $O(V^3)$

总结

表 6.4　BFS 算法、Dijkstra 算法和 Floyd 算法求最短路径的总结

	BFS 算法	Dijkstra 算法	Floyd 算法
用途	求单源最短路径	求单源最短路径	求各顶点之间的最短路径
无权图	适用	适用	适用
带权图	不适用	适用	适用
带负权值的图	不适用	不适用	适用
带负权回路的图	不适用	不适用	不适用
时间复杂度	$O(\|V\|^2)$或 $O(\|V\|+\|E\|)$	$O(\|V\|^2)$	$O(\|V\|^3)$

有向无环图

若一个有向图中不存在环，则称为有向无环图，简称DAG图

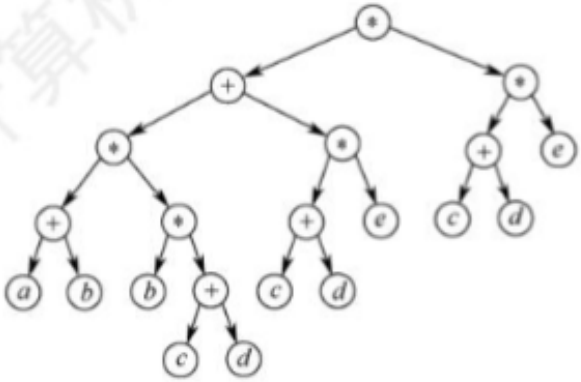


图 6.20　二叉树描述表达式

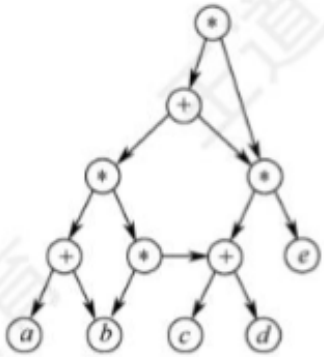


图 6.21　有向无环图描述表达式

拓扑排序

拓扑排序是对有向无环图的顶点的一种排序，它使得若存在一条从顶点A到顶点B的路径，则在排序中B出现在A的后面。每个AOV网都有一个或多个拓扑排序序列

拓扑排序和回路的关系

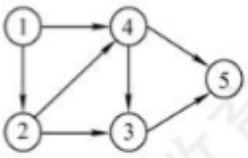
对一个 AOV 网进行拓扑排序的算法有很多，下面介绍比较常用的一种方法的步骤：

① 从 AOV 网中选择一个没有前驱（入度为 0）的顶点并输出。

② 从网中删除该顶点和所有以它为起点的有向边。

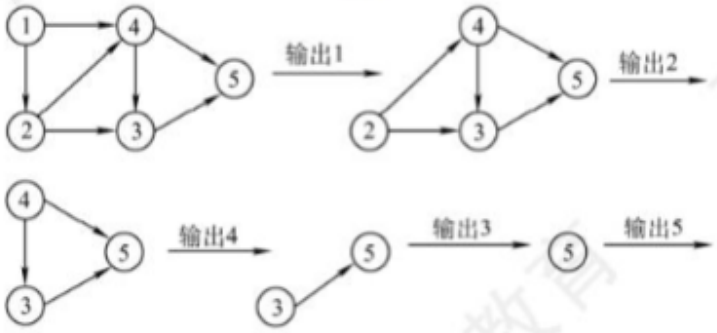
③ 重复①和②直到当前的 AOV 网为空或当前网中不存在无前驱的顶点为止。后一种情况说明有向图中必然存在环。

拓扑排序的实例



结点号	1	2	3	4	5
初始入度	0	1	2	2	2
第 1 轮		0	2	1	2
第 2 轮			1	0	2
第 3 轮			0		1
第 4 轮					0
第 5 轮					

(a)



(b)

图 6.22　有向无环图的拓扑排序过程

拓扑排序时间复杂度

邻接表 $O(V+E)$,邻接矩阵 $O(V^2)$

逆拓扑排序

- 选择一个没有后继的顶点
- 从网中删除该顶点以及以它为终点的有向边
- 重复1，2直至为空

拓扑序列的存在性和唯一性

- 有多个直接后继不唯一，只有唯一的，则唯一
- 邻接矩阵是三角矩阵的一般有拓扑序列

关键路径

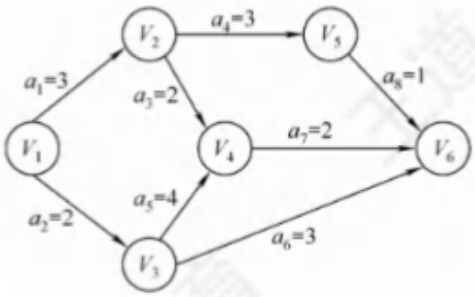
- AOE网中最大路径长度为关键路径，上的活动是关键活动。

参量

- 事件 v_k 最早发生时间 $v_c(K)$
  - $v_c(K)=Max\{v_c(j)+Weight(v_j,v_k)\}$

- 最迟发生时间 $v_l(K)$
  - $v_l(k) = Min\{v_l(j) - Weight(v_k, v_j)\}$
- 活动 a_i 最早发生时间 $e(i)$ 在边 $< v_k, v_j >$上
  - $e(i) = v_c(k)$
- 活动 a_i 最迟发声时间 $l(i)$
  - $l(i) = v_l(j) - Weight(v_k, v_j)$
- a_i 的最迟开始时间和最早发声的时间的差 $d(i)$
  - 等于零则为关键活动

求解过程



	v_1	v_2	v_3	v_4	v_5	v_6
$v_c(i)$	0	3	2	6	6	8
$v_l(i)$	0	4	2	6	7	8

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
$e(i)$	0	0	3	3	2	2	6	6
$l(i)$	1	0	4	4	2	5	6	7
$l(i) - e(i)$	1	0	1	1	0	3	0	1

图 6.23　求解关键路径的过程

总结

表 6.5　采用不同存储结构时各种图算法的时间复杂度

	Dijkstra	Floyd	Prim	Kruskal	DFS	BFS	拓扑排序	关键路径
邻接矩阵	$O(n^2)$	$O(n^3)$	$O(n^2)$	-	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
邻接表	-	-	-	$O(e\log e)$	$O(n+e)$	$O(n+e)$	$O(n+e)$	$O(n+e)$