图

目 录

Contents

图

1 图的定义

一个图(G)定义为一个偶对(V,E)，记为G=(V,E)。其中：V是顶点(Vertex)的非空有限集合，记为V(G)；E是无序集V&V的一个子集，记为E(G)，其元素是图的弧(Arc)。用|V|表示图G中的顶点个数，也称为图G的阶；用|E|表示图G中的条数。

将顶点集合为空的图称为空图。

弧(Arc)：表示两个顶点v和w之间存在一个关系，用顶点偶对<v,w>表示。通常根据图的顶点偶对将图分为有向图和无向图。

图

有向图(Digraph)： 若图G的关系集合E(G)中，顶点偶对<v,w>的v和w之间是有序的，称图G是有向图，v称为弧尾，w称为弧头。

无向图(Undigraph)： 若图G的关系集合E(G)中，顶点偶对(v,w)的v和w之间是无序的，称图G是无向图。

图

例1：设有有向图G1和无向图G2，形式化定义分别是：

G1=(V1 ， E1)

V1={a,b,c,d,e}

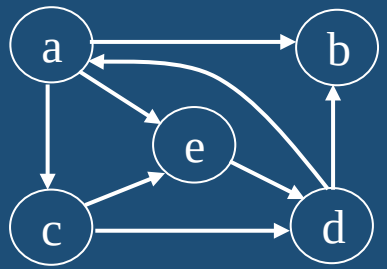
E1={<a,b>,<a,c>, <a,e>,<c,d>,<c,e> ,<d,a>,<d,b>,<e,d>}

G2=(V2 ， E2)

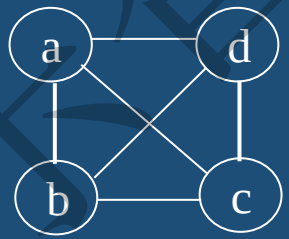
V2={a,b,c,d}

E2={(a,b), (a,c), (a,d), (b,d), (b,c), (c,d)}

它们所对应的图如图7-1所示。



(a) 有向图G1



(b) 无向图G2

图

简单图：一个图G如果满足（1）不存在重复边（2）不存在顶点到自身的边，则称图G为简单图。

完全图：任意两个顶点之间都存在边。含有n个顶点的无向完全图有n(n-1)/2条边。在有向图中，任意两个顶点之间都存在方向相反的两条弧，则称为有向完全图。含有n个顶点的有向完全图有n(n-1)条有向边。

完全无向图：对于无向图G=(V，E)，若$\forall v_i$，$v_j \in V$，当$v_i \neq v_j$时，有$(v_i, v_j) \in E$，即图中任意两个不同的顶点间都有一条无向边，这样的无向图称为完全无向图。

完全有向图：对于有向图G=(V，E)，若$\forall vi$，$vj \in V$，当$vi \neq vj$时，有$<vi, vj> \in E \wedge <vj, vi> \in E$，即图中任意两个不同的顶点间都有一条弧，这样的有向图称为完全有向图。

图

有很少边或弧的图（e<nlogn）的图称为稀疏图，反之称为稠密图。

权(Weight)：与图的边和弧相关的数。权可以表示从一个顶点到另一个顶点的距离或耗费。

路径：顶点V1到顶点V2之间的一条路径是指顶点序列，V1，Vil，...V2。路径上边的数目称为路径长度。第一个顶点和最后一个顶点相同的路径称为环或者回路。如果一个图有n个顶点，并且有大于n-1条边，则此图一定有环。

图

简单路径：在路径序列中，顶点不重复出现的路径称为简单路径。

简单回路：除第一个顶点和最后一个顶点之外，其余顶点不重复出现的回路。

连通：在无向图中，若从顶点v到顶点w有路径存在，则称v和w是连通的。

连通图：若图G中任意两个顶点都是连通的，则图G称为连通图，否则称为非连通图。

子图和生成子图：设有图G=(V，E)和G’=(V’，E’)，若V’⊂V且E’⊂E，则称图G’是G的子图；若V’=V且E’⊂E，则称图G’是G的一个生成子图。

图

连通分量：无向图G的的极大连通子图称为G的连通分量。任何连通图的连通分量只有一个，即是其自身，非连通的无向图有多个连通分量。如果一个图有n个顶点，并且有小于n-1条边，则此图必是非连通图。其中，**极大即要求该连通子图包含其所有的边。**

强连通图：在有向图中，若从顶点v到顶点w和从顶点w到顶点v之间都有路径则称这两个顶点是强连通的。若图中任意一对顶点都是强连通的，则此图为强连通图。

强连通分量：在有向图中的极大强连通子图称为有向图的强连通分量。

图

顶点的邻接(Adjacent)：对于无向图G=(V，E)，若边(v,w)∈E，则称顶点v和w 互为邻接点，即v和w相邻接。

顶点的度、入度、出度：对于无向图G=(V，E)，∀vi∈V，图G中依附于vi的边的数目称为顶点vi的度(degree)，记为TD(vi)。

显然，在无向图中，所有顶点度的和是图中边的2倍。即 $\sum TD(vi)=2e$ i=1, 2, …, n ，e为图的边数。（度和公式）

对有向图G=(V，E)，若∀vi ∈V，图G中以vi作为起点的有向边(弧)的数目称为顶点vi的出度(Outdegree)，记为OD(vi)；以vi作为终点的有向边(弧)的数目称为顶点vi的入度(Indegree)，记为ID(vi)。顶点vi的出度与入度之和称为vi的度，记为TD(vi)。即

TD(vi)=OD(vi)+ID(vi)

图的存储

图的常用的存储结构有：

邻接矩阵、邻接链表、邻接多重表和边表。

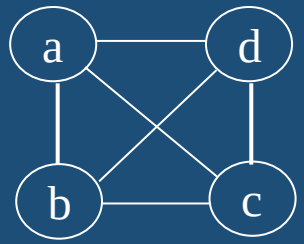
在这里只涉及邻接矩阵（顺序存储结构）、邻接链表（链接存储结构）。

1　邻接矩阵(数组)表示法

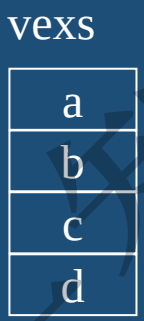
基本思想：对于有n个顶点的图，用一维数组vexs[n]存储顶点信息，用二维数组A[n][n]存储顶点之间关系的信息。该二维数组称为邻接矩阵。在邻接矩阵中，以顶点在vexs数组中的下标代表顶点，邻接矩阵中的元素A[i][j]存放的是顶点i到顶点j之间关系的信息。

1. 无向图的数组表示

(1) 无权图的邻接矩阵



(a) 无向图

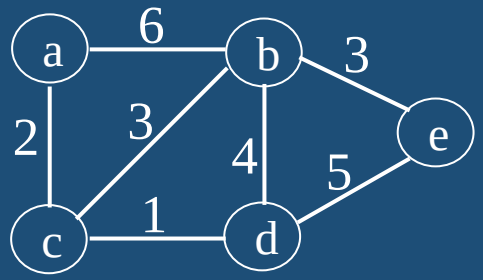


(b) 顶点矩阵

	a	b	c	d
a	0	1	1	1
b	1	0	1	1
c	1	1	0	1
d	1	1	1	0

(c) 邻接矩阵

(2) 带权图的邻接矩阵



(a) 带权无向图

vexs

a
b
c
d
e

(b) 顶点矩阵

	a	b	c	d	e
a	∞	6	2	∞	∞
b	6	∞	3	4	3
c	2	3	∞	1	∞
d	∞	4	3	∞	5
e	∞	3	∞	5	∞

(c) 邻接矩阵

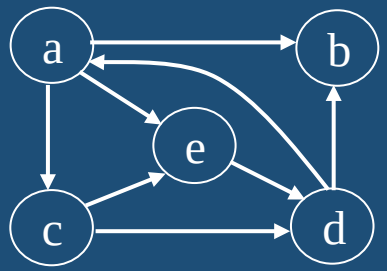
（3）无向图邻接矩阵的特性(重点)

- 邻接矩阵是对称方阵；
- 对于顶点vi，其度数是第i行的非0元素的个数；
- 无向图的边数是上(或下)三角形矩阵中非0元素个数。

图的存储

2. 有向图的数组表示

(1) 无权图的邻接矩阵



(a) 有向图

vexs

a
b
c
d
e

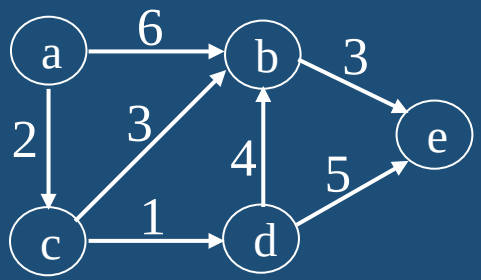
(b) 顶点矩阵

	a	b	c	d	e
a	0	1	1	0	1
b	0	0	0	0	0
c	0	0	0	1	1
d	1	1	0	0	0
e	0	0	0	1	0

(c) 邻接矩阵

图的存储

(2) 带权图的邻接矩阵



(a) 带权有向图

vexs

a
b
c
d
e

(b) 顶点矩阵

	a	b	c	d	e
a	∞	6	2	∞	∞
b	∞	∞	∞	∞	3
c	∞	3	∞	1	∞
d	∞	4	∞	∞	5
e	∞	∞	∞	∞	∞

(c) 邻接矩阵

图的存储

(3) 有向图邻接矩阵的特性（重点）

- 对于顶点v_i，第i行的非0元素的个数是其出度$OD(v_i)$；第i列的非0元素的个数是其入度$ID(v_i)$。
- 邻接矩阵中非0元素的个数就是图的弧的数目。

图的存储

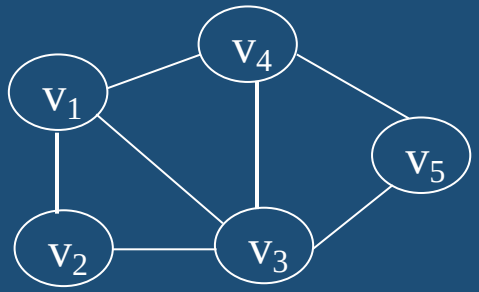
2　邻接链表法

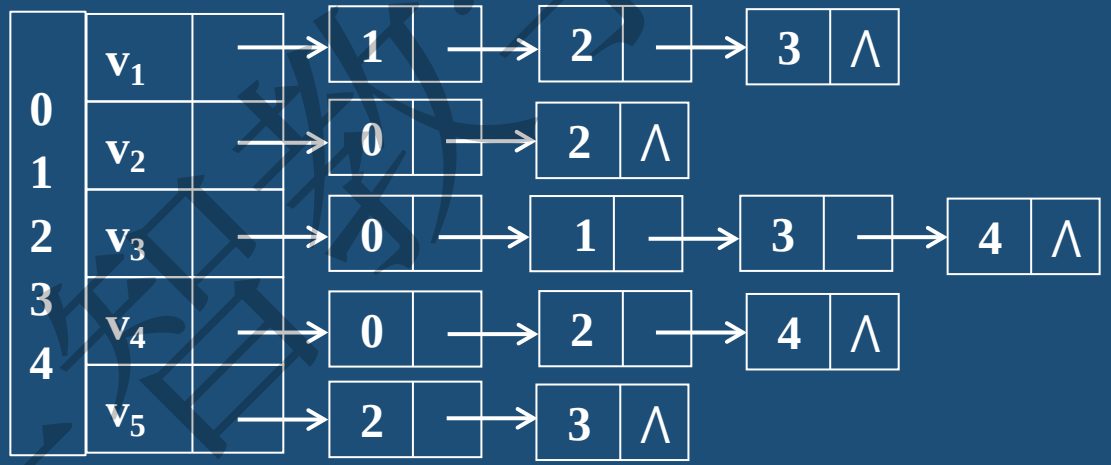
基本思想：对图的每个顶点建立一个单链表，存储该顶点所有邻接顶点及其相关信息。每一个单链表设一个表头结点。

第i个单链表表示依附于顶点V_i的边(对有向图是以顶点V_i为头或尾的弧)。

图的存储

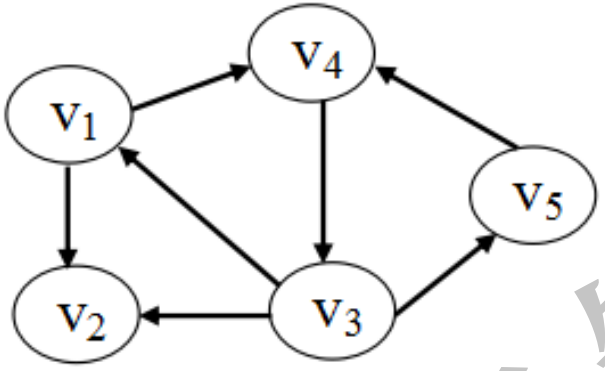
1. 无向图的邻接表



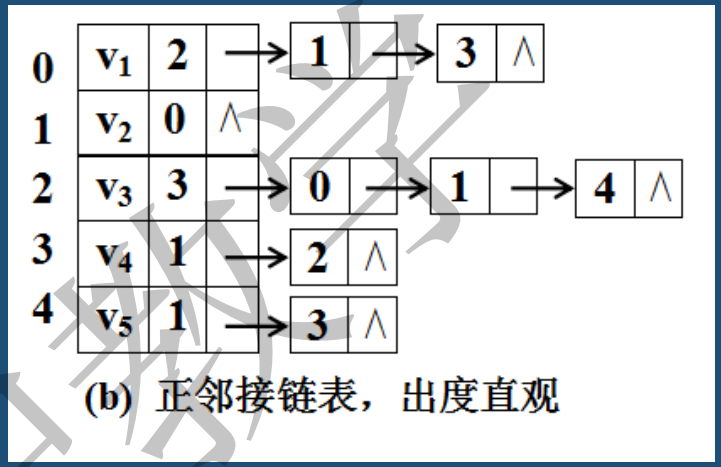


图的存储

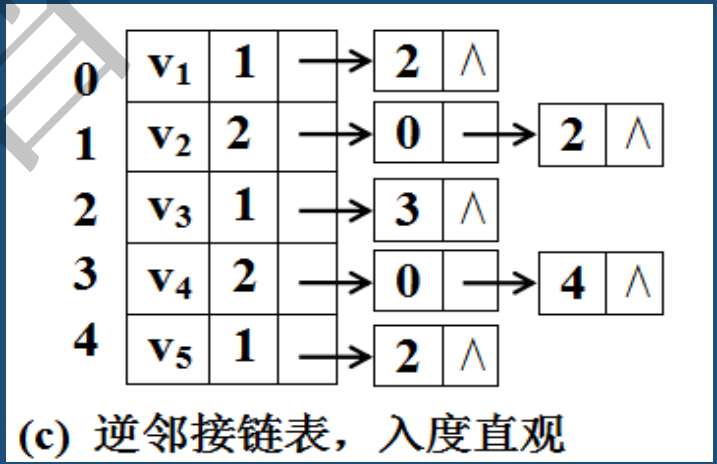
2. 有向图的邻接表



(a) 有向图



(b) 正邻接链表，出度直观



(c) 逆邻接链表，入度直观

图的存储

3. 邻接表法的特点

◆ 表头向量中每个分量就是一个单链表的头结点，分量个数就是图中的顶点数目；

◆ 在边或弧稀疏的条件下，用邻接表表示比用邻接矩阵表示节省存储空间；

◆ 在无向图，顶点V_i的度是第i个链表的结点数；

◆ 对有向图可以建立正邻接表或逆邻接表。正邻接表是以顶点V_i为出度(即为弧的起点)而建立的邻接表；逆邻接表是以顶点V_i为入度(即为弧的终点)而建立的邻接表；

◆ 在有向图中，第i个链表中的结点数是顶点V_i的出(或入)度；求入 (或出)度，须遍历整个邻接表；

◆ 在邻接表上容易找出任一顶点的第一个邻接点和下一个邻接点；

图的遍历

图的遍历(Travering Graph)：从图的某一顶点出发，访遍图中的其余顶点，且每个顶点仅被访问一次。图的遍历算法是各种图的操作的基础。

图的遍历算法有深度优先搜索算法和广度优先搜索算法。

图的遍历

深度优先搜索(Depth First Search--DFS)遍历类似树的先序遍历，是树的先序遍历的推广。

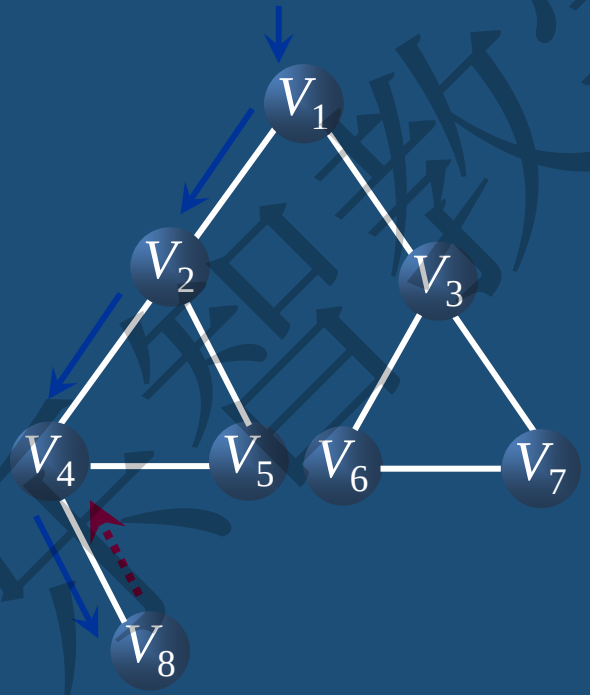
图的深度优先遍历借助栈来实现。

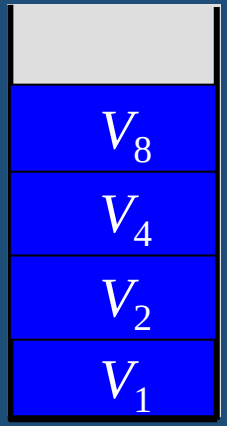
DFS算法是一个递归算法，需要借助一个递归工作栈，故它的空间复杂度为O(V)

遍历图的过程实质上是对每个顶点查找其邻接点，其消耗的时间取决于所采用的存储结构。当以邻接矩阵表示时，查找每个结点的邻接点所需时间O(V)，总的时间复杂度为$O(V^2)$。当以邻接表表示时，查找所有顶点的邻接点所需时间为O(e)，访问顶点所需时间为O(v)，此时，总的时间复杂度O(e+v)

深一层递归

递归返回

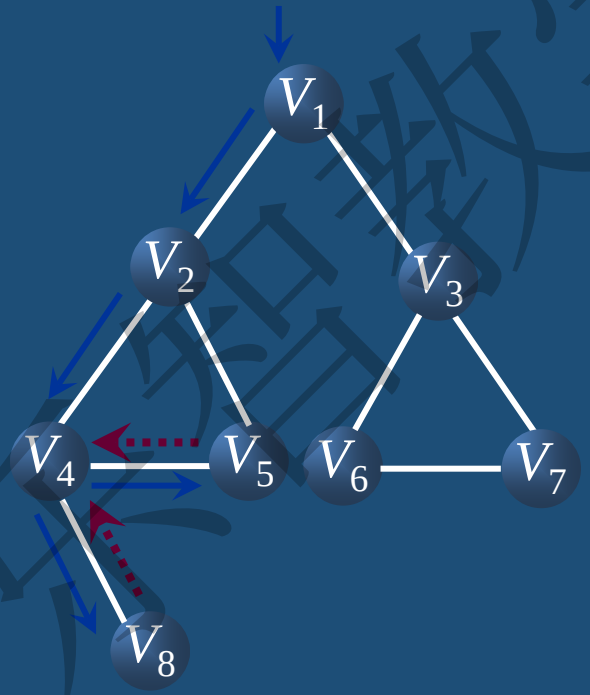


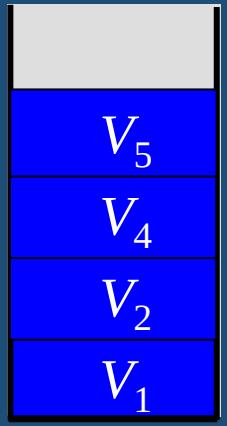


遍历序列：V_1 V_2 V_4 V_8

深一层递归

递归返回

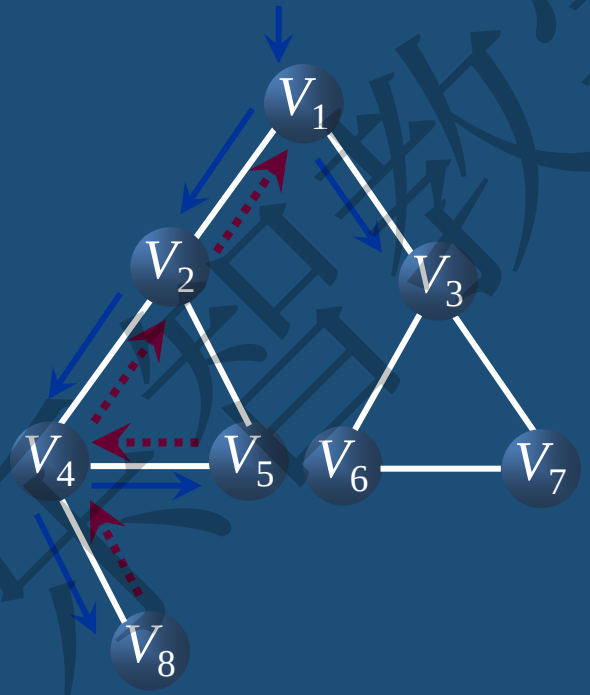


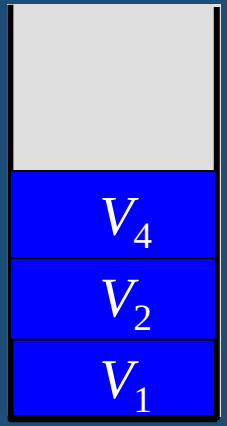


遍历序列：V_1 V_2 V_4 V_8 V_5

深一层递归

递归返回

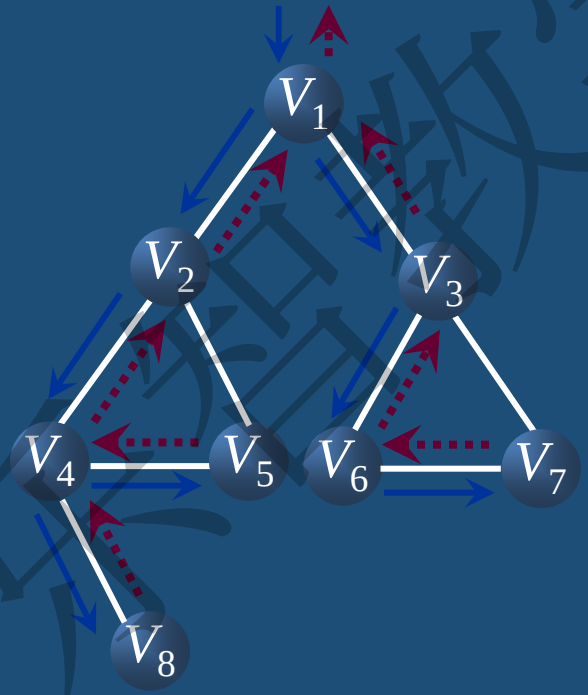


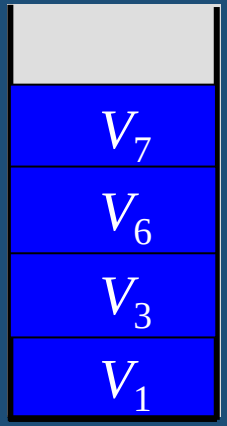


遍历序列：V_1 V_2 V_4 V_8 V_5

深一层递归

递归返回





遍历序列：V_1 V_2 V_4 V_8 V_5 V_3 V_6 V_7

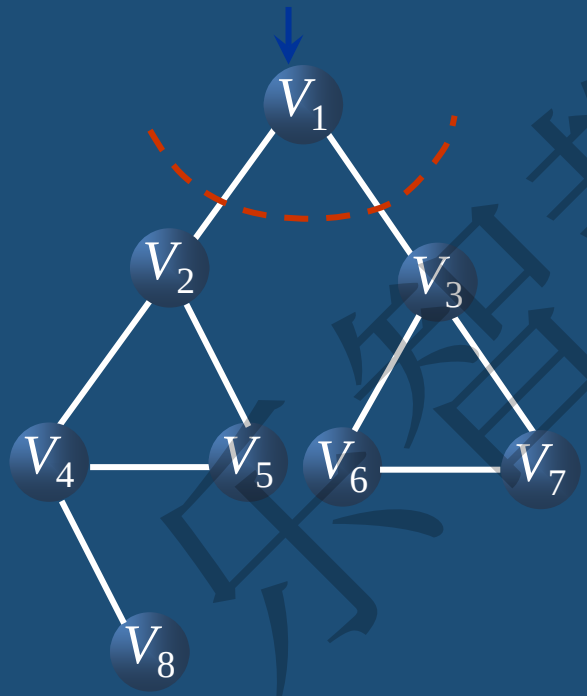
图的遍历

广度优先搜索(Breadth First Search--BFS)遍历类似树的按层次遍历的过程。

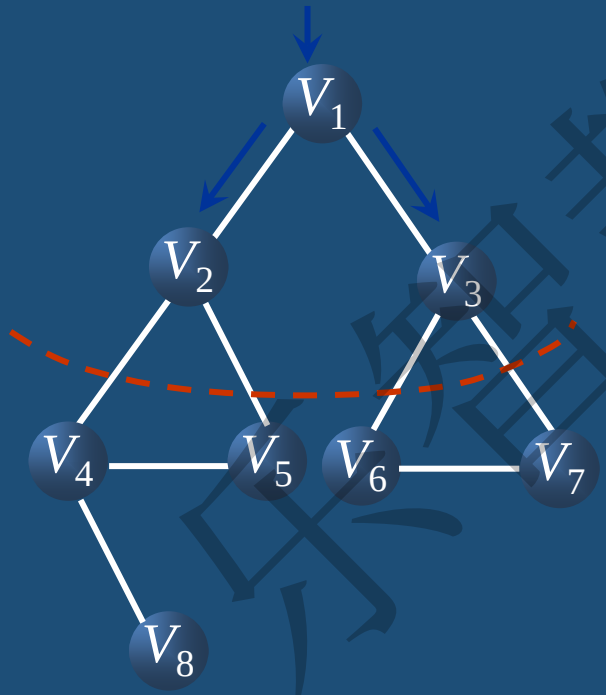
图的深度优先遍历借助队列来实现。

BFS算法需要借助一个队列，故它的空间复杂度为O(V)

当以邻接矩阵表示时，查找每个结点的邻接点所需时间O(V)，总的时间复杂度为$O(V^2)$。当以邻接表表示时，访问顶点所需时间为O(v)，查找所有顶点的邻接点时，每条边至少访问一次，所需时间为O(e)，此时，总的时间复杂度O(e+v)

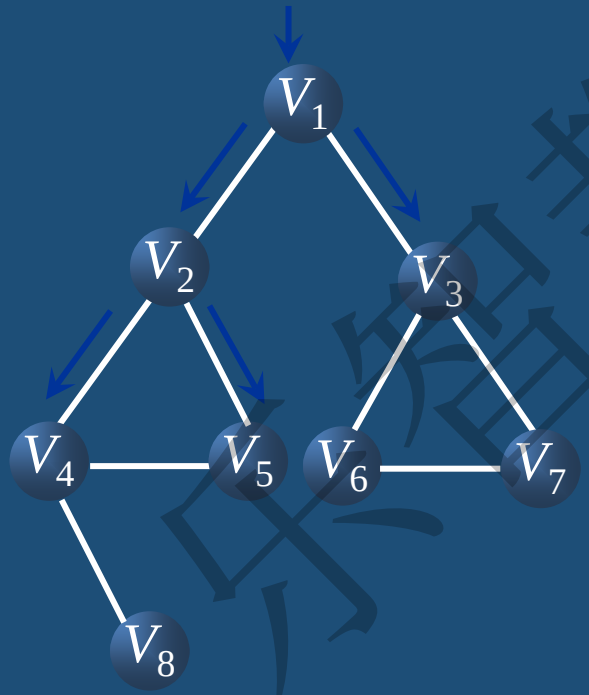


遍历序列：V_1



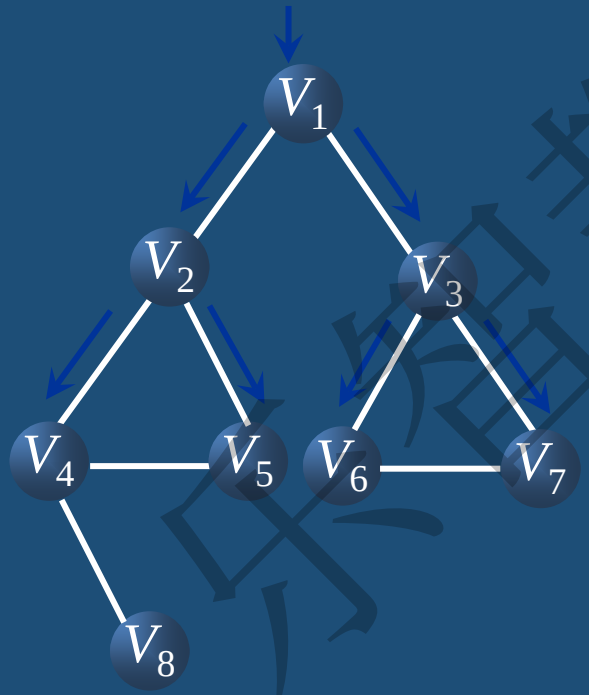
V_2 V_3

遍历序列：V_1 V_2 V_3



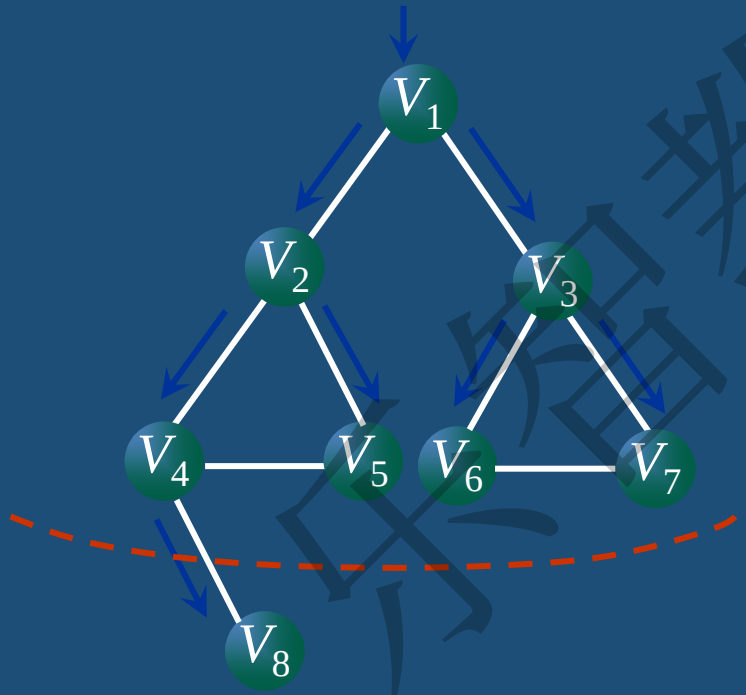
$V_3\ V_4\ V_5$

遍历序列：$V_1\ V_2\ V_3\ V_4\ V_5$



$V_4\ V_5\ V_6\ V_7$

遍历序列：$V_1\ V_2\ V_3\ V_4\ V_5\ V_6\ V_7$



$V_5\ V_6\ V_7\ V_8$

遍历序列：$V_1\ V_2\ V_3\ V_4\ V_5\ V_6\ V_7\ V_8$

图的应用

1　最小生成树

如果连通图是一个带权图，则其生成树中的边也带权，生成树中所有边的权值之和称为生成树的代价。

最小生成树(Minimum Spanning Tree)：带权连通图中代价最小的生成树称为最小生成树。

构造最小生成树的算法有许多，基本原则是：

- 尽可能选取权值最小的边，但不能构成回路；
- 选择n-1条边构成最小生成树。

图的应用

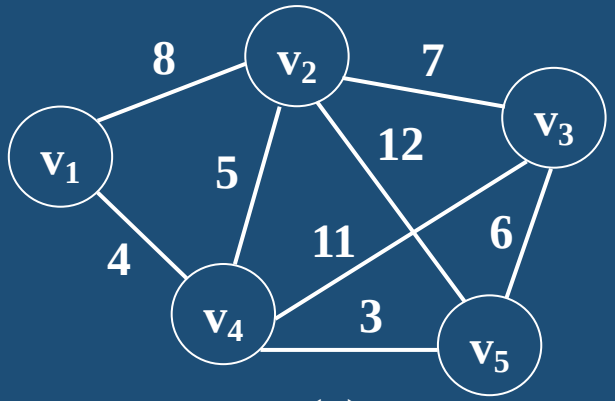
1.普里姆(Prim)算法

从连通网$G=(V, E)$中找最小生成树$T=(U, TE)$。

基本思想：设$G=(V, E)$是具有n个顶点的连通网，$T=(U, TE)$是G的最小生成树，T的初始状态为$U=\{u_0\}$（$u_0 \in V$），$TE=\{\ \}$，重复执行下述操作：在所有$u \in U$，$v \in V-U$的边中找一条代价最小的边(u, v)并入集合TE，同时v并入U，直至$U=V$。

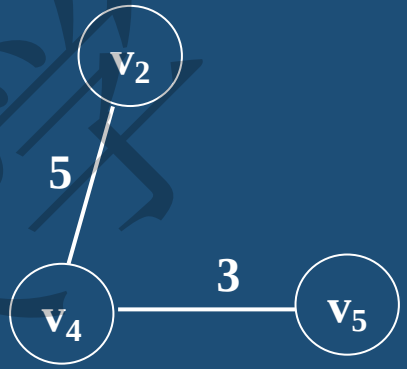
整个算法的时间复杂度是$O(n^2)$，与边的数目无关，适用于求解稠密的图最小生成树。

图的应用

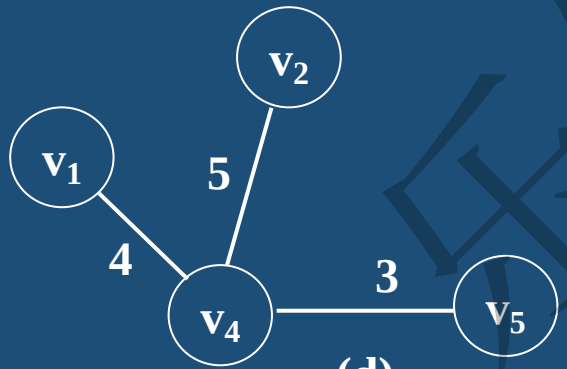


(a)

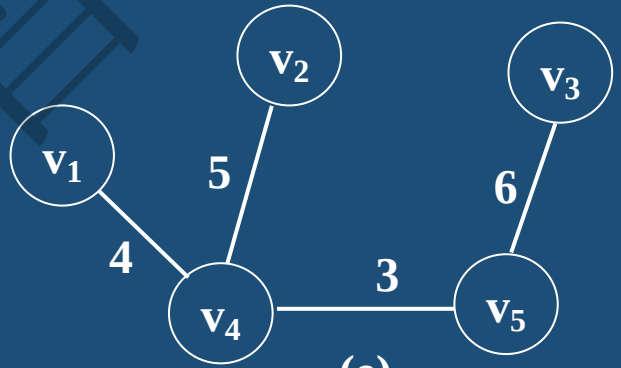
(b)



(c)



(d)



(e)

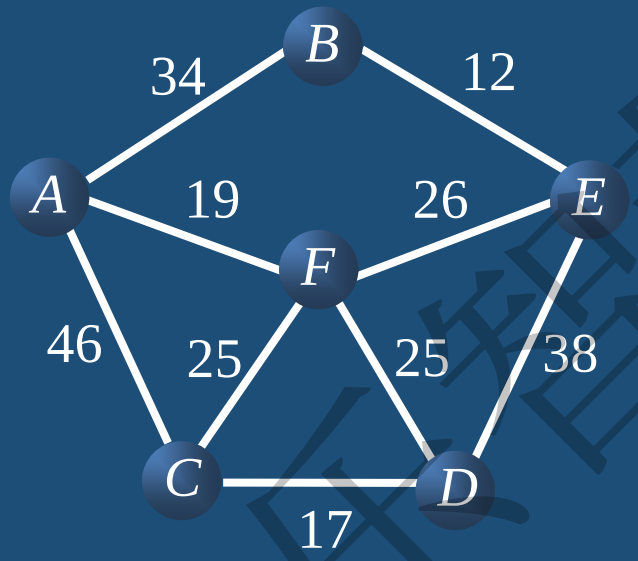
图的应用

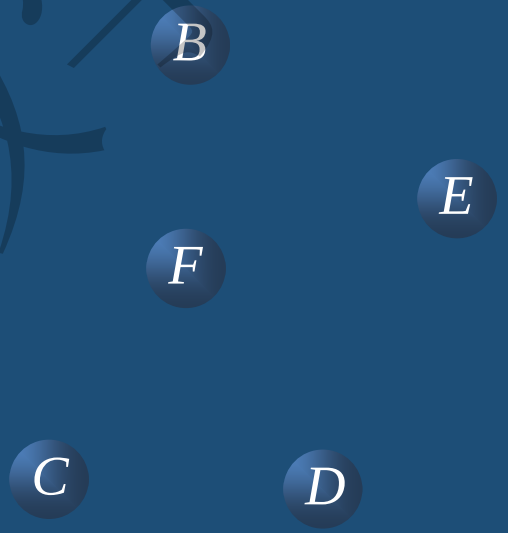
2.克鲁斯卡尔(Kruskal)算法

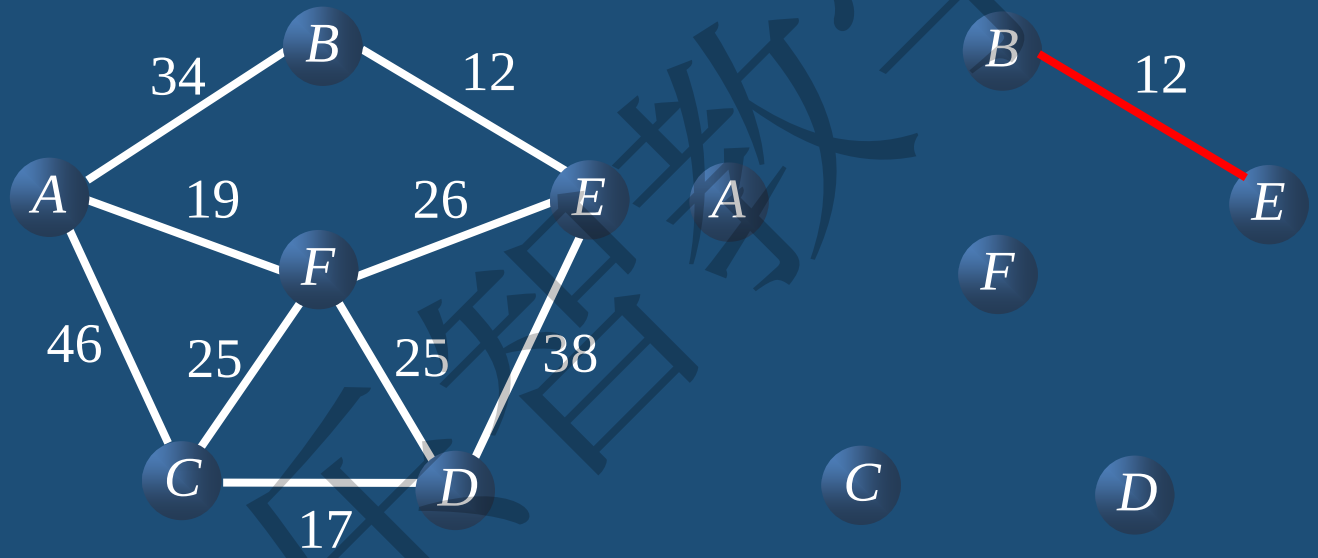
从连通网$G=(V, E)$中找最小生成树$T=(U, TE)$。

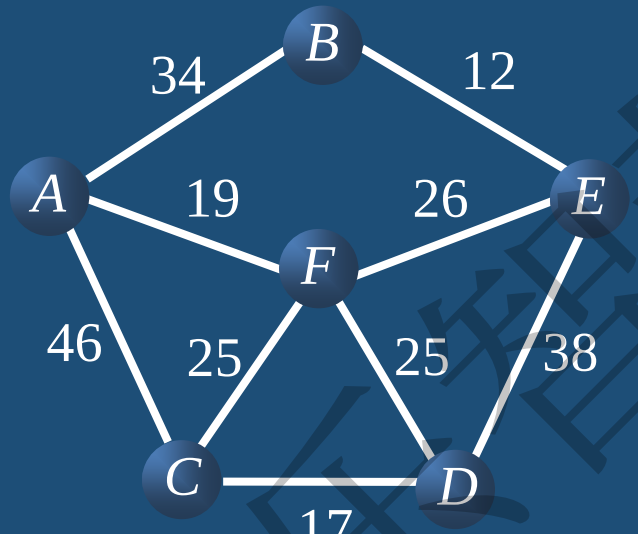
基本思想：设无向连通网为$G=(V, E)$，令G的最小生成树为$T=(U, TE)$，其初态为$U=V$，$TE=\{\ \}$，然后，按照边的权值由小到大的顺序，考察G的边集E中的各条边。若被考察的边的两个顶点属于T的两个不同的连通分量，则将此边作为最小生成树的边加入到T中，同时把两个连通分量连接为一个连通分量；若被考察边的两个顶点属于同一个连通分量，则舍去此边，以免造成回路，如此下去，当T中的连通分量个数为1时，此连通分量便为G的一棵最小生成树。

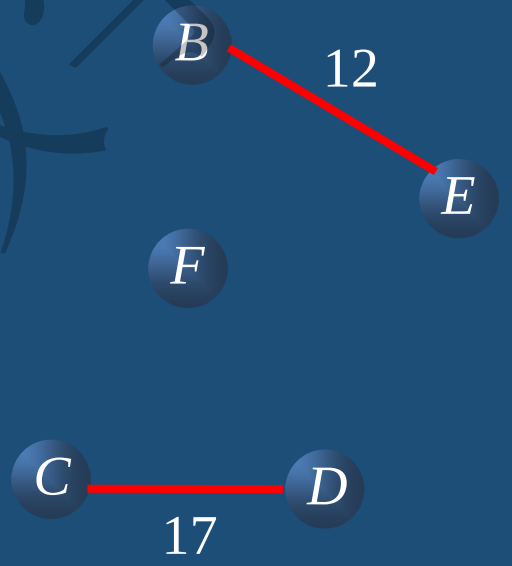
整个算法的时间复杂度是O(eloge) 。适用于边稀疏而顶点较多的图。

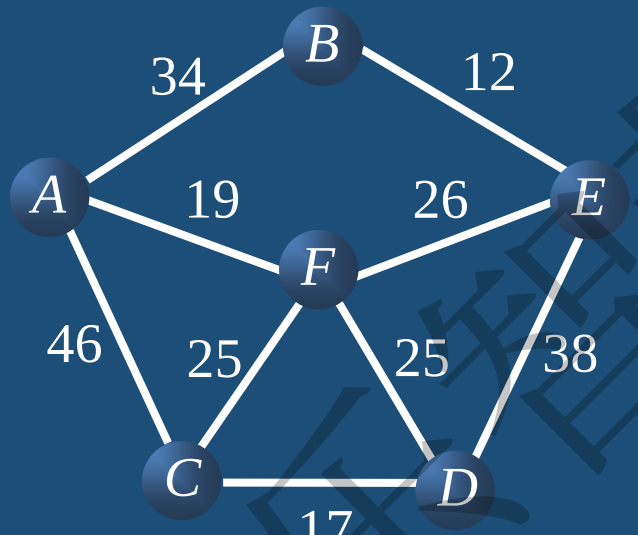


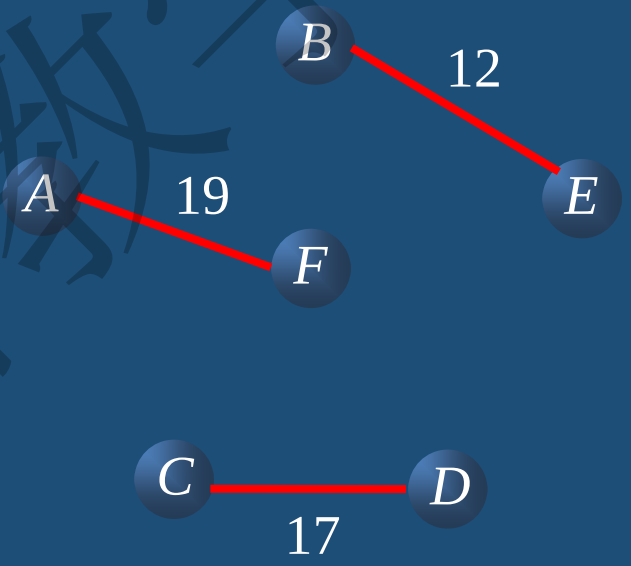


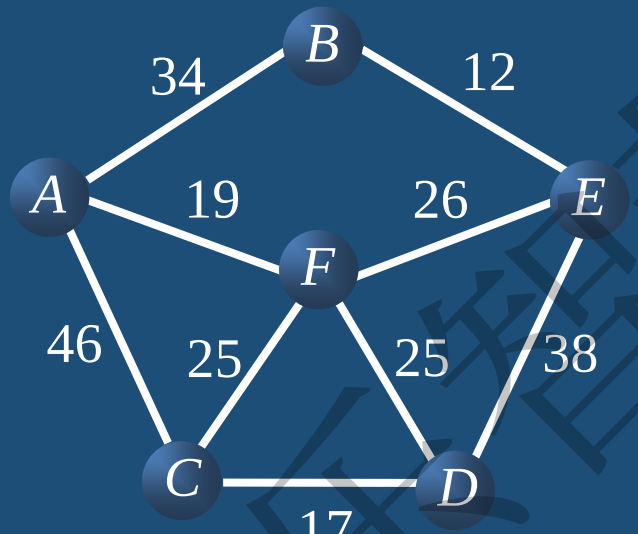


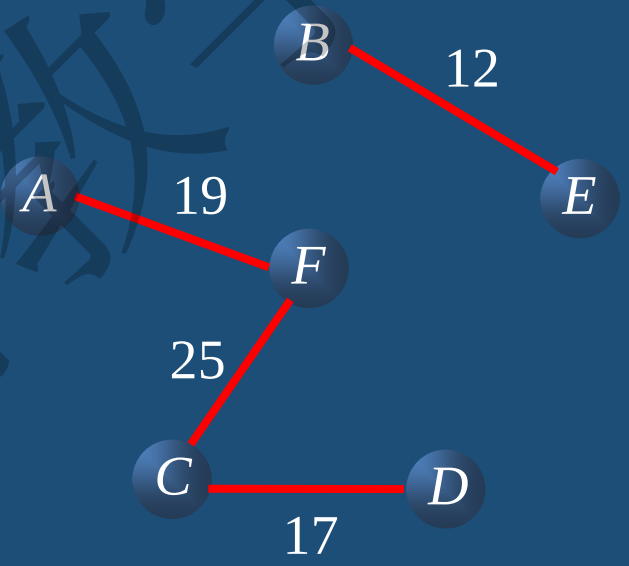


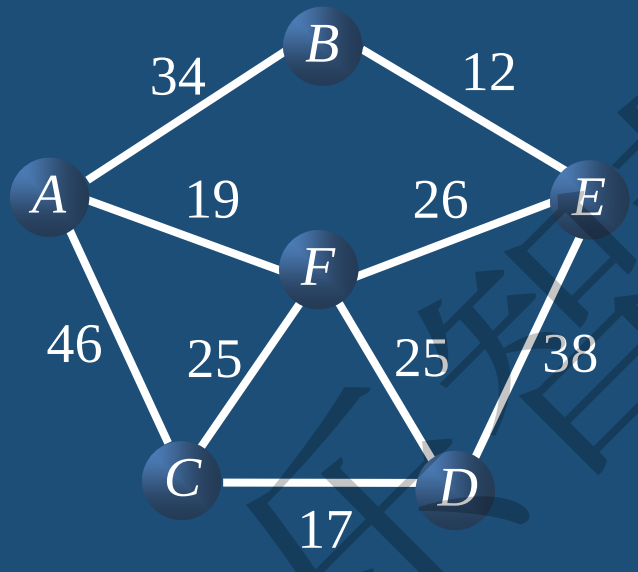


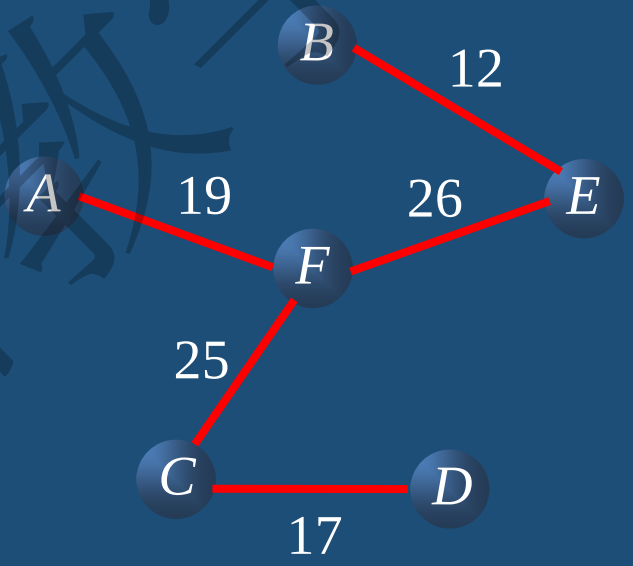












图的应用

2 最短路径

若用带权图表示交通网，图中顶点表示地点，边代表两地之间有直接道路，边上的权值表示路程(或所花费用或时间)。从一个地方到另一个地方的路径长度表示该路径上各边的权值之和。问题：

- 两地之间是否有通路?
- 在有多条通路的情况下，哪条最短?

考虑到交通网的有向性，直接讨论的是带权有向图的最短路径问题，但解决问题的算法也适用于无向图。

将一个路径的起始顶点称为源点，最后一个顶点称为终点。

图的应用

1.单源点最短路径

对于给定的有向图G=(V，E)及单个源点V_s，求V_s到G的其余各顶点的最短路径。

针对单源点的最短路径问题，Dijkstra提出了一种按路径长度递增次序产生最短路径的算法，即迪杰斯特拉(Dijkstra)算法。

在构造过程中设置了两个辅助数组。

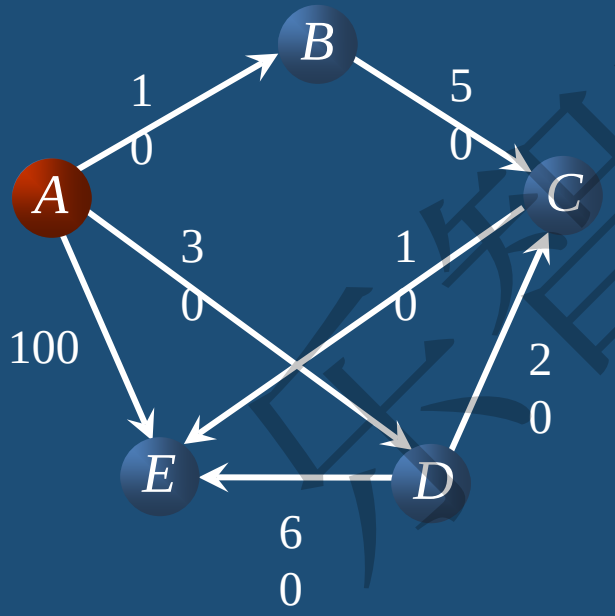
图的应用

基本思想

从图的给定源点到其它各个顶点之间客观上应存在一条最短路径，在这组最短路径中，按其长度的递增次序，依次求出到不同顶点的最短路径和路径长度。

即按长度递增的次序生成各顶点的最短路径，即先求出长度最小的一条最短路径，然后求出长度第二小的最短路径，依此类推，直到求出长度最长的最短路径。整个算法的时间复杂度是$O(n^2)$。

Dijkstra算法



S={A}

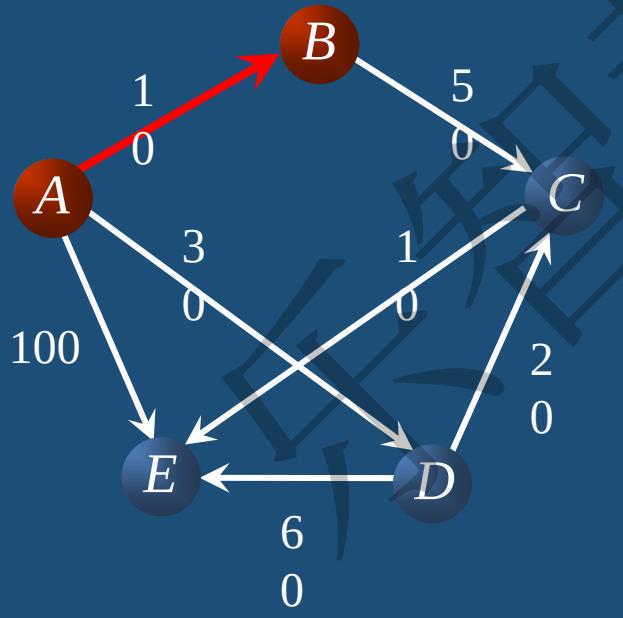
A→B:(A, B)10

A→C:(A, C)∞

A→D: (A, D)30

A→E: (A, E)100

Dijkstra算法



S={A, B}

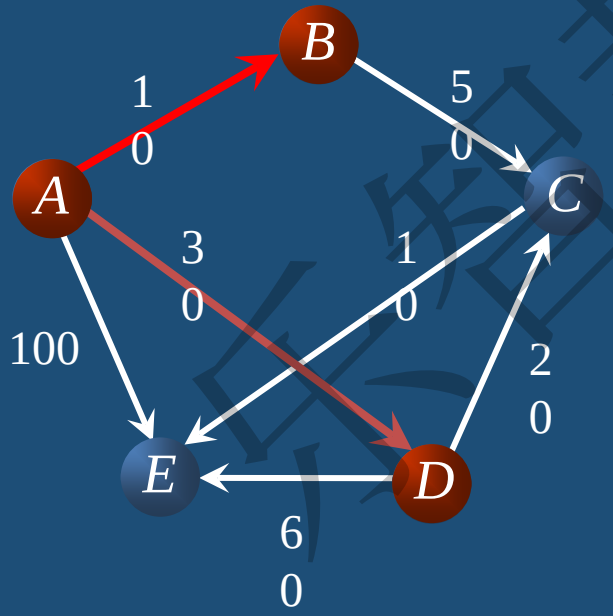
A→B:(A, B)10

A→C:(A, B, C)60

A→D: (A, D)30

A→E: (A, E)100

Dijkstra算法



S={A, B, D}

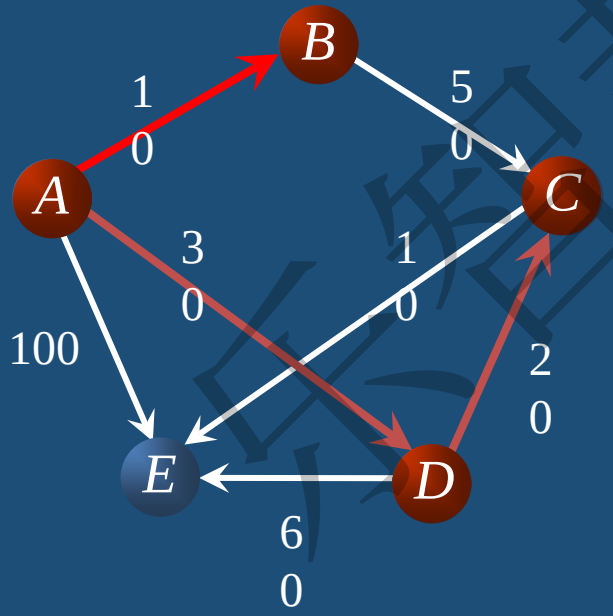
A→B:(A, B)10

A→C:(A, D, C)50

A→D: (A, D)30

A→E: (A, D, E)90

Dijkstra算法



S={A, B, D, C}

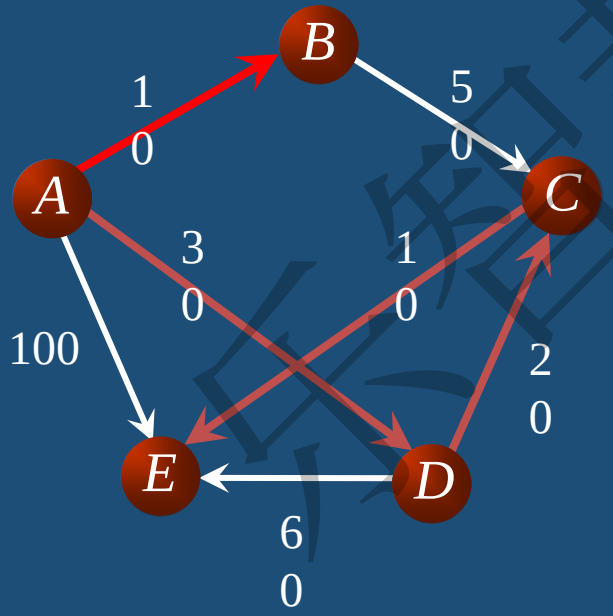
A→B:(A, B)10

A→C:(A, D, C)50

A→D: (A, D)30

A→E: (A, D, C, E)60

Dijkstra算法



S={A, B, D, C, E}

A→B:(A, B)10

A→C:(A, D, C)50

A→D: (A, D)30

A→E: (A, D, C, E)60

图的应用

2.每一对顶点之间的最短路径

问题描述：给定带权有向图$G=(V, E)$，对任意顶点$v_i, v_j \in V$（$i \neq j$），求顶点v_i到顶点v_j的最短路径。

解决办法1：每次以一个顶点为源点，调用Dijkstra算法n次。显然，时间复杂度为$O(n^3)$。

解决办法2：弗洛伊德提出的求每一对顶点之间的最短路径算法——Floyd算法，其时间复杂度也是$O(n^3)$，但形式上要简单些。

图的应用

Floyd算法

基本思想：对于从v_i到v_j的弧，进行n次试探：首先考虑路径v_i, v_0, v_j是否存在，如果存在，则比较v_i, v_j和v_i, v_0, v_j的路径长度，取较短者为从v_i到v_j的中间顶点的序号不大于0的最短路径。在路径上再增加一个顶点v_1，依此类推，在经过n次比较后，最后求得的必是从顶点v_i到顶点v_j的最短路径。

图的应用

AOV网：在一个表示工程的有向图中，用顶点表示活动，用弧表示活动之间的优先关系，称这样的有向图为顶点表示活动的网，简称AOV网。

AOV网特点：

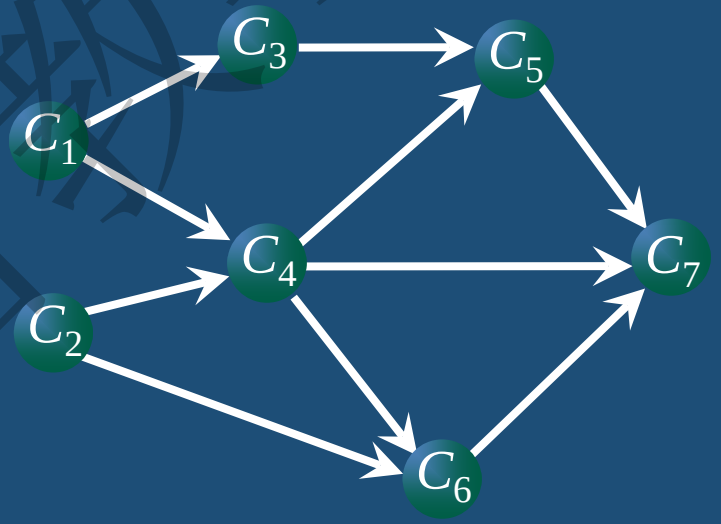
1.AOV网中的弧表示活动之间存在的某种制约关系。

2.AOV网中不能出现回路 。

图的应用

AOV网

编号	课程名称	先修课
C_1	高等数学	无
C_2	计算机导论	无
C_3	离散数学	C_1
C_4	程序设计	C_1，C_2
C_5	数据结构	C_3，C_4
C_6	计算机原理	C_2，C_4
C_7	数据库原理	C_4，C_5，C_6



图的应用

拓扑排序

拓扑序列：设$G=(V, E)$是一个具有n个顶点的有向图，V中的顶点序列$v_1, v_2, \ldots, v_n$称为一个拓扑序列，当且仅当满足下列条件：若从顶点v_i到v_j有一条路径，则在顶点序列中顶点v_i必在顶点v_j之前。

拓扑排序：对一个有向图构造拓扑序列的过程称为**拓扑排序**。

算法的时间复杂度是**O(n+e)**。

拓扑序列使得AOV网中所有应存在的前驱和后继关系都能得到满足。

图的应用

拓扑排序

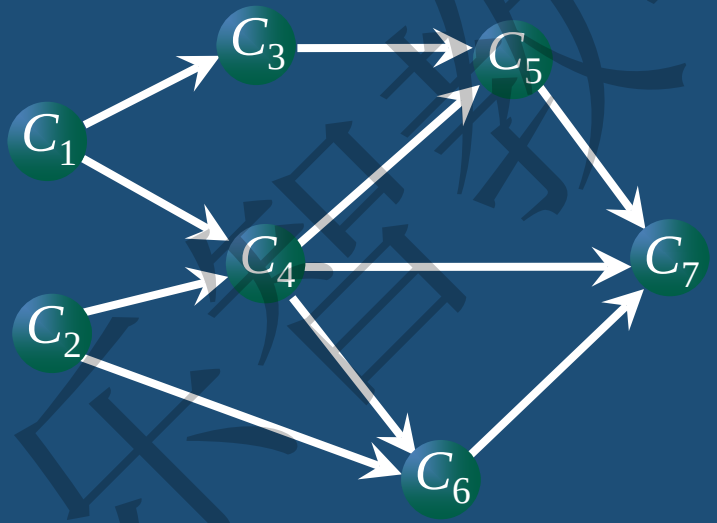
基本思想：

(1) 从AOV网中选择一个没有前驱的顶点并且输出；

(2) 从AOV网中删去该顶点，并且删去所有以该顶点为尾的弧；

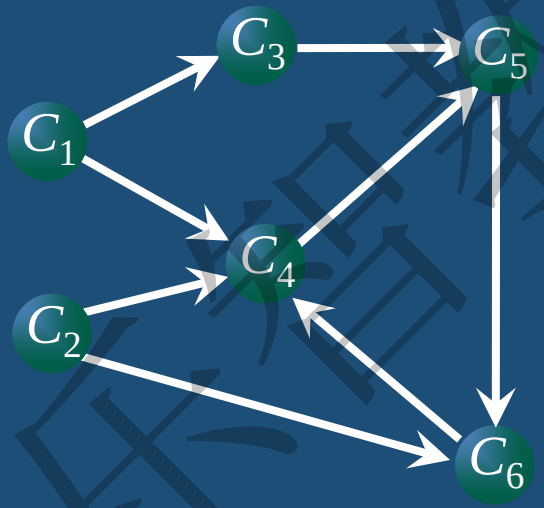
(3) 重复上述两步，直到全部顶点都被输出，或AOV网中不存在没有前驱的顶点。

拓扑排序



拓扑序列：C_1, C_2, C_3, C_4, C_5, C_6, C_7

拓扑排序



说明AOV网中存在回路。

拓扑序列：　C_1，　C_2，C_3，

图的应用

AOE网

AOE网：在一个表示工程的带权有向图中，用顶点表示事件，用有向边表示活动，边上的权值表示活动的持续时间，称这样的有向图叫做**边表示活动**的网，简称AOE网。AOE网中没有入边的顶点称为**始点**（或源点），没有出边的顶点称为**终点**（或汇点）。

AOE网的性质：

⑴ 只有在某顶点所代表的事件发生后，从该顶点出发的各活动才能开始；

⑵ 只有在进入某顶点的各活动都结束，该顶点所代表的事件才能发生。

图的应用

关键路径

关键路径：在AOE网中，从始点到终点具有最大路径长度（该路径上的各个活动所持续的时间之和）的路径称为关键路径。

关键活动：关键路径上的活动称为关键活动。

由于AOE网中的某些活动能够同时进行，故完成整个工程所必须花费的时间应该为始点到终点的最大路径长度。关键路径长度是整个工程所需的最短工期。

关键路径

要找出关键路径，必须找出关键活动，即不按期完成就会使整个工程完成的时间延长。

首先计算以下与关键活动有关的量：

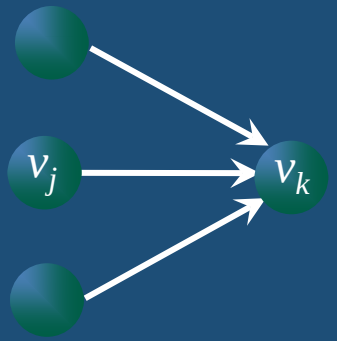
(1) 事件的最早发生时间ve[k]
(2) 事件的最迟发生时间vl[k]
(3) 活动的最早开始时间e[i]
(4) 活动的最晚开始时间l[i]

最后计算各个活动的时间余量 l[k] - e[k]，时间余量为0者即为关键活动。整个算法的时间复杂度是O(n+e) 。

图的应用

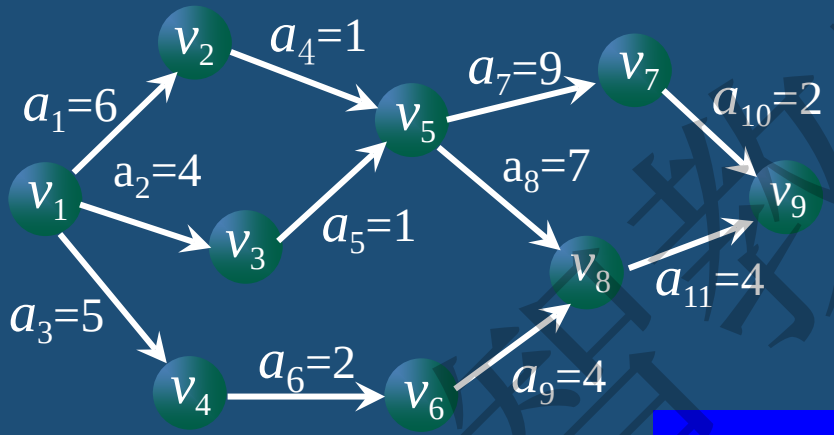
(1) 事件的最早发生时间ve[k]

ve[k]是指从始点开始到顶点v_k的最大路径长度。这个长度决定了所有从顶点v_k发出的活动能够开工的最早时间。



$$\begin{cases} ve[1]=0 \\ ve[k]=\max\{ve[j]+len<v_j, v_k>\}\ (<v_j, v_k>\in p[k]) \end{cases}$$

p[k]表示所有到达vk的有向边的集合

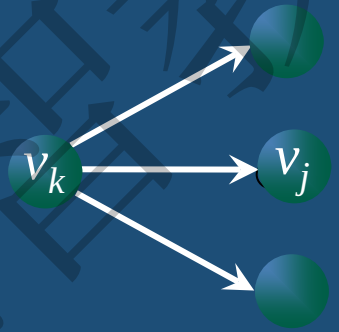


ve[k]=max{ve[j]+len<v_j, v_k>}

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
ve[k]	0	6	4	5	7	7	16	14	18

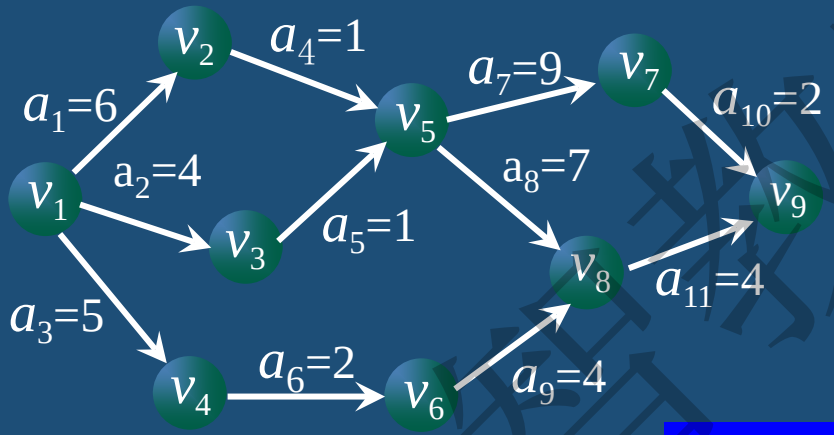
(2) 事件的最迟发生时间vl[k]

vl[k]是指在不推迟整个工期的前提下,事件v_k允许的最晚发生时间。



$$\begin{cases} vl[n]=ve[n] \\ vl[k]=\min\{vl[j]-len<v_k, v_j>\}\ (<v_k, v_j>\in s[k]) \end{cases}$$

s[k]为所有从v_k发出的有向边的集合



vl[k]=min{vl[j]-len<v_k, v_j>}

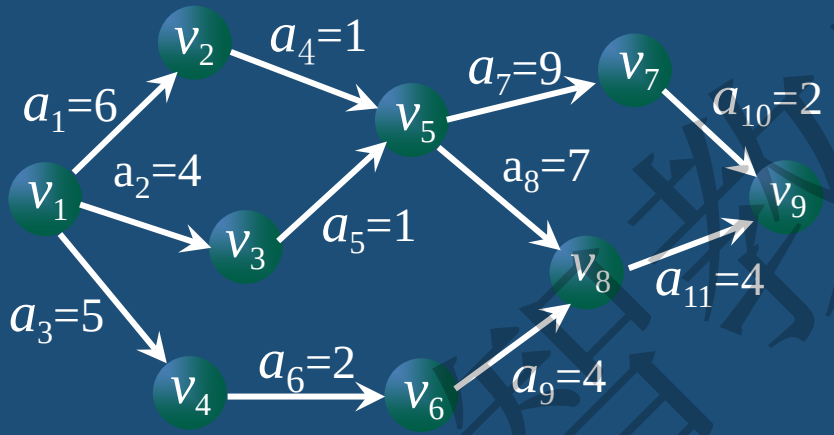
	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
ve[k]	0	6	4	5	7	7	16	14	18
vl[k]	0	6	6	8	7	10	16	14	18

(3) 活动的最早开始时间e[i]

若活动a_i是由弧<v_k，v_j>表示，则活动a_i的最早开始时间应等于事件v_k的最早发生时间。因此，有：

e[i]=ve[k]

某一个活动的最早开始时间是该活动的弧尾顶点事件的最早发生时间



e[i]=ve[k]

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
ve[k]	0	6	4	5	7	7	16	14	18

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
e[i]	0	0	0	6	4	5	7	7	7	16	14

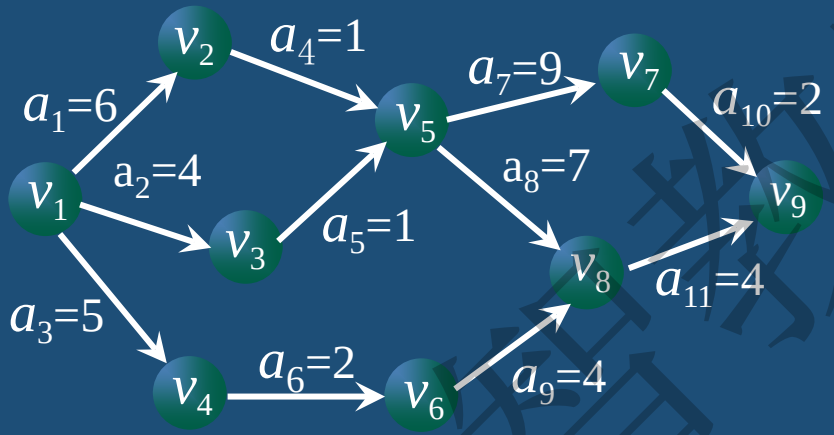
(4) 活动的最晚开始时间l[i]

活动a_i的最晚开始时间是指，在不推迟整个工期的前提下，a_i必须开始的最晚时间。

若a_i由弧<v_k，v_j>表示，则a_i的最晚开始时间要保证事件v_j的最迟发生时间不拖后。因此，有：

l[i]=vl[j]-len<v_k, v_j>

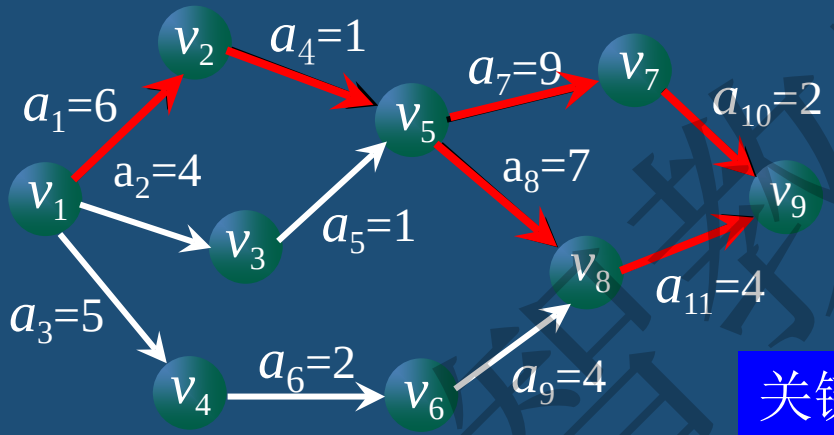
对于某一活动的最晚开始时间是它的弧头顶点事件允许的最晚发生时间减去该活动所持续的时间。



l[i]=vl[j]-len<v_k, v_j>

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
vl[k]	0	6	6	8	7	10	16	14	18

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
l[i]	0	2	3	6	6	8	7	7	10	16	14



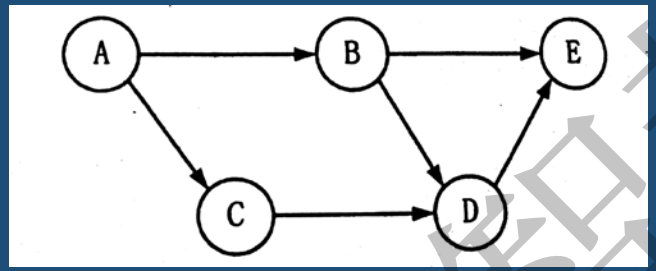
关键活动：l[i]=e[i]的活动

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
e[i]	0	0	0	6	4	5	7	7	7	16	14
l[i]	0	2	3	6	6	8	7	7	10	16	14

图（真题检测）

1.在AOE网中，从源点到汇点各活动时间总和最长的路径称为（ ）

2.对下图进行拓扑排序，可以得到（ ）个不同的拓扑序列。



A.1　　B.2　　C.3　　D.4

答案：关键路径　　B

图（真题检测）

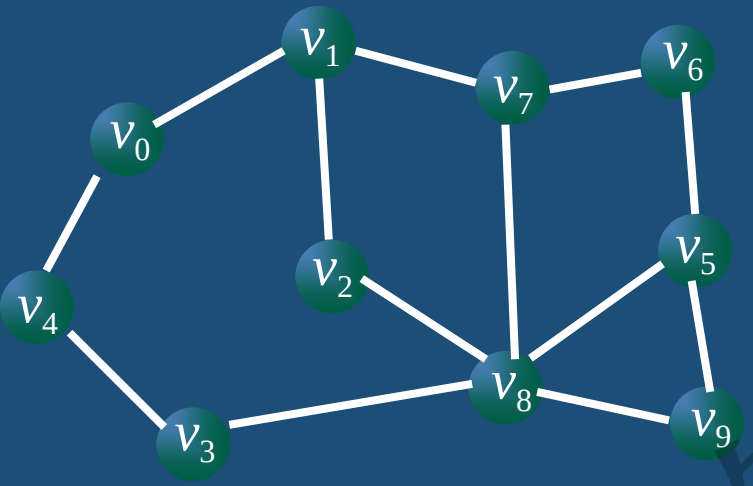
3.已知一个无向图的顶点集合V和边集E如下：

V={v0,v1,v2,v3,v4,v5,v6,v7,v8,v9}
E={(v0,v1),(v0,v4),(v1,v2),(v1,v7),(v2,v8),(v3,v4),(v3,v8),(v5,v6),(v5,v8),(v5,v9),(v6,v7),(v7,v8),(v8,v9)}

① 写出该图的邻接矩阵。

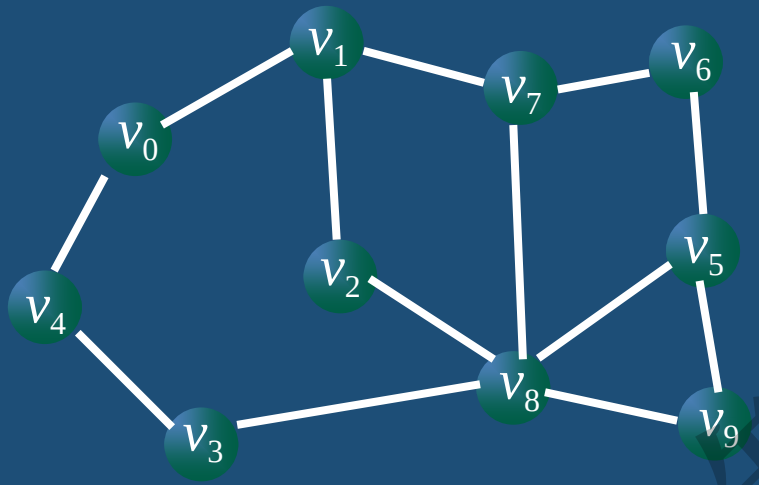
② 当该图用邻接矩阵表示时，分别写出从顶点v0出发按深度优先遍历搜索得到的顶点序列和按广度优先遍历搜索得到的顶点序列。

图（真题检测）



	V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9
V_0	0	1	0	0	1	0	0	0	0	0
V_1	1	0	1	0	0	0	0	1	0	0
V_2	0	1	0	0	0	0	0	0	1	0
V_3	0	0	0	0	1	0	0	0	1	0
V_4	1	0	0	1	0	0	0	0	0	0
V_5	0	0	0	0	0	0	1	0	1	1
V_6	0	0	0	0	0	1	0	1	0	0
V_7	0	1	0	0	0	0	1	0	1	0
V_8	0	0	1	1	0	1	0	1	0	1
V_9	0	0	0	0	0	1	0	0	1	0

图（真题检测）



深度优先遍历搜索：

V_0 V_4 V_3 V_8 V_9 V_5 V_6 V_7 V_1 V_2

V_0 V_1 V_2 V_8 V_7 V_6 V_5 V_9 V_3 V_4

深度优先遍历搜索：

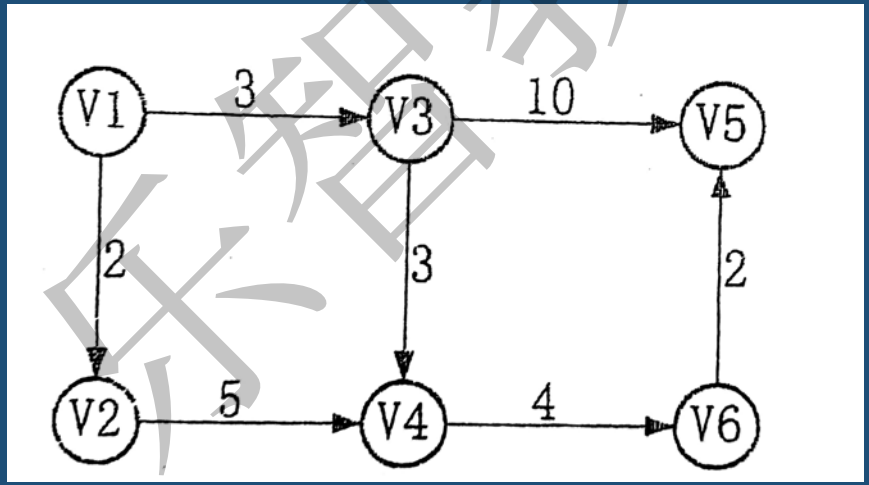
V_0 V_4 V_1 V_3 V_2 V_7 V_8 V_6 V_9 V_5

注意：图的深度优先遍历和广度优先遍历都不止一个，考生只需写出符合题意的一个即可！！！

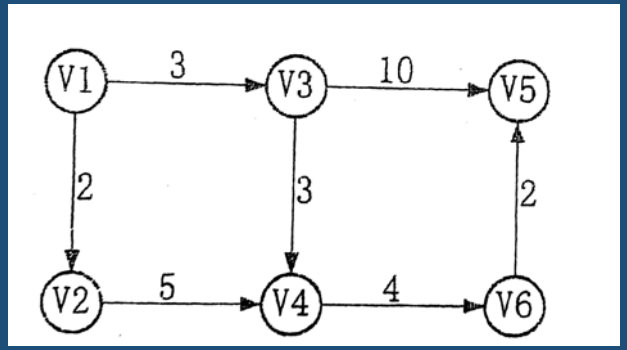
图（真题检测）

4.如下图所示：

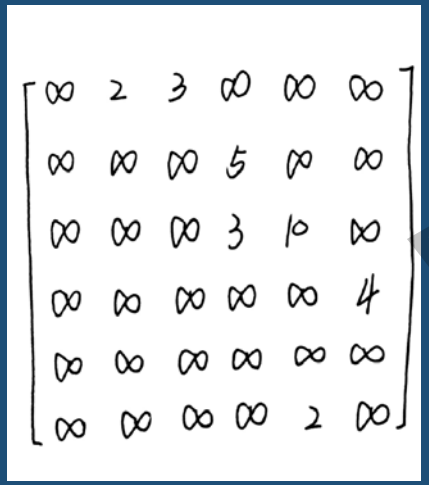
① 写出该图的邻接矩阵。
② 写出全部拓扑排序结果。
③ 用Dijkstra算法求从V1顶点到其他各顶点的最短距离和最短路径。



图（真题检测）



① 写出该图的邻接矩阵。
② 写出全部拓扑排序结果。

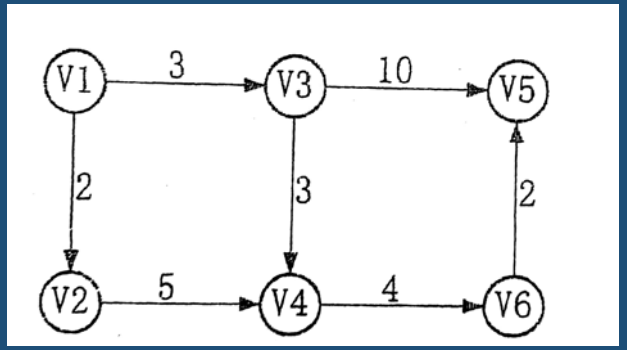


全部拓扑排序结果如下：

$V_1\ V_2\ V_3\ V_4\ V_6\ V_5$

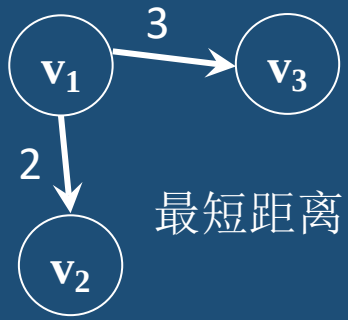
$V_1\ V_3\ V_2\ V_4\ V_6\ V_5$

图（真题检测）



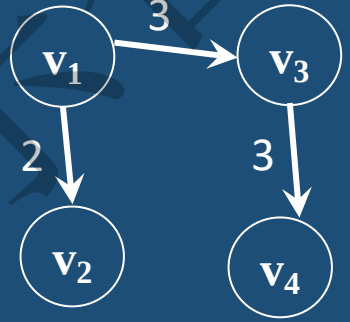
用Dijkstra算法求从V1顶点到其他各顶点的最短距离和最短路径。

最短距离：0



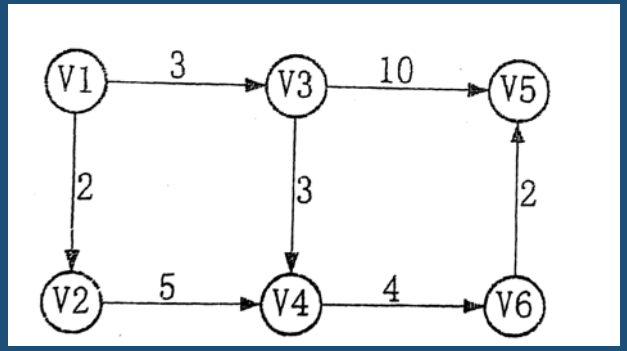
最短距离：5

最短距离：2

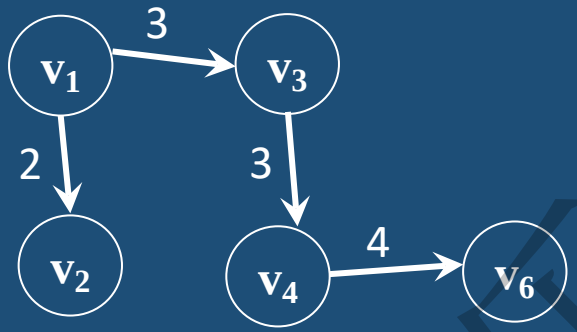


最短距离：8

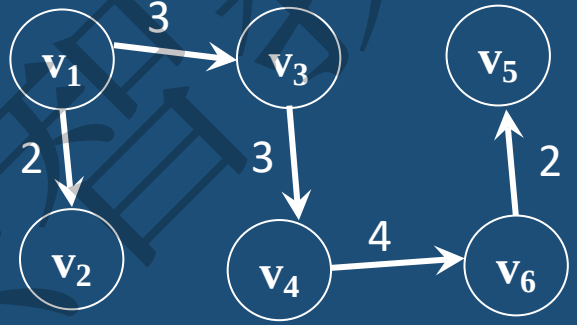
图（真题检测）



用Dijkstra算法求从V1顶点到其他各顶点的最短距离和最短路径。



最短距离：12



最短距离：14

谢 谢 观 看