王道考研——数据结构

WWW.CSKAOYAN.COM

第五章 图

本章总览

王道考研/CSKAOYAN.COM

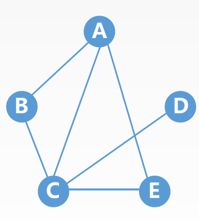
本节内容

图

图的基本概念

图的基本概念

图 图G由顶点集V和边集E组成，记为G=（V, E），
其中V(G)表示图G中顶点的有限非空集；E(G)表示图G中顶点之间的关系（边）集合
|V| 表示图G中顶点的个数，也称图G的阶；|E| 表示图G中边的条数



G=（V, E)

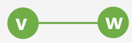
V={A, B, C, D, E}　　|V|=5

E={(A, B), (A, C), (A, E), (B, C), (C, D), (C, E)}　|E|=6

★ 线性表，树都可以为空，但图不能为空

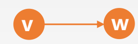
图的基本概念

无向图 & 有向图

无向边 

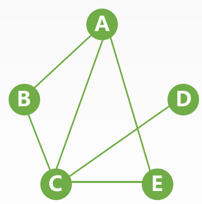
无序对（v, w）=（w，v）

v，w互为邻接点

有向边 

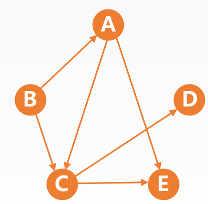
有序对　<v, w>

v邻接到w，或w邻接自v



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

E={(A, B),(A, C),(A, E),(B, C),(C, D),(C, E)}



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

E={<B, A>,<A, C>,<A, E>,<B, C>,<C, D>,<C, E>}

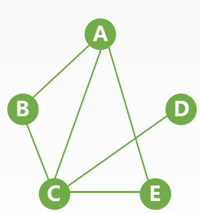
王道考研/CSKAOYAN. COM

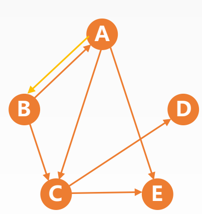
图的基本概念

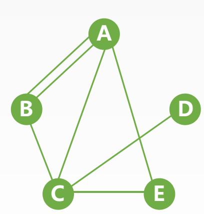
简单图 & 多重图

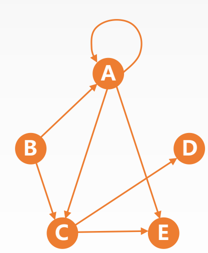
无重复边，
不存在结点到自身的边

存在重复边，
或存在结点到自身的边









王道考研/CSKAOYAN. COM

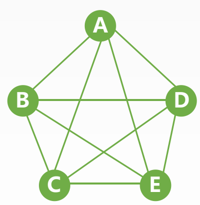
图的基本概念

完全图

无向完全图

任意两个顶点之间都存在边

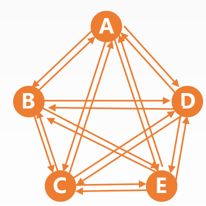
n个顶点有n(n-1)/2边



有向完全图

任意两个顶点之间都存在方向相反的弧

n个顶点有n(n-1)边

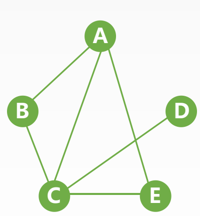


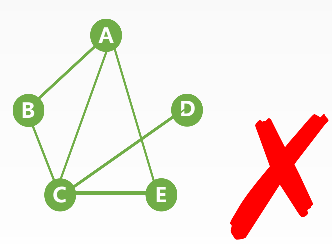
图的基本概念

子图

设有两个图G=(V, E) 和G'=(V', E'), 若V'是V的子集，且E'是E的子集，
则称G'为G的子图，

且若V(G) = V(G')则称G'为G的生成子图



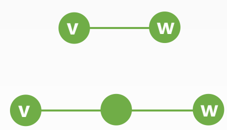


图的基本概念

无向图

连通

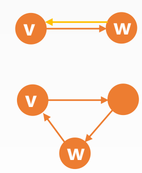
若从顶点v到顶点w有路径存在，则称v和w是连通



有向图

强连通

若从顶点v到顶点w和顶点w到顶点v都有路径存在，则称v和w是强连通



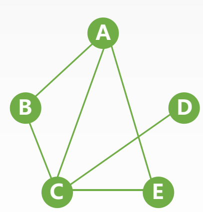
王道考研/CSKAOYAN.COM

图的基本概念

无向图

连通图

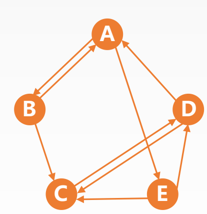
任意两个结点之间都是连通的



有向图

强连通图

任意两个结点之间都是强连通的



王道考研/CSKAOYAN.COM

图的基本概念

无向图

连通图

任意两个结点之间都是连通的

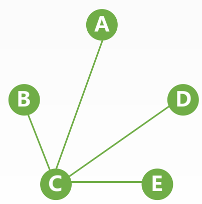
有向图

强连通图

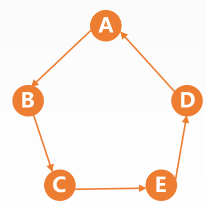
任意两个结点之间都是强连通的

? n个顶点的连通图（强连通图）最少有多少条边

n-1



n



图的基本概念

无向图

连通分量

极大连通子图

有向图

强连通分量

极大强连通子图

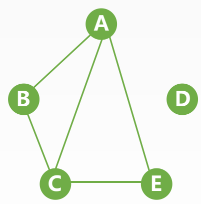
对于G的一个（强）连通子图G'，如果不存在G的另一个（强）连通子图G''，使得 G' ⊂ G'' ,则称G'为G的（强）连通分量。

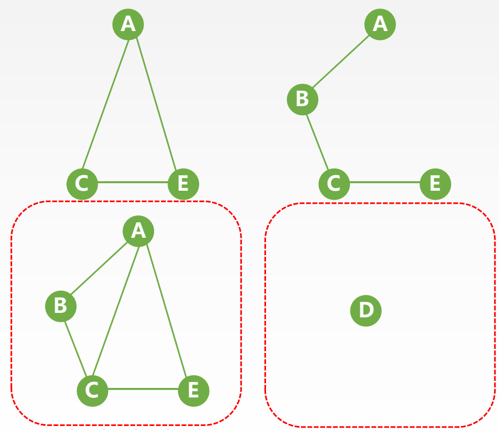
图的基本概念

无向图

连通分量

极大连通子图



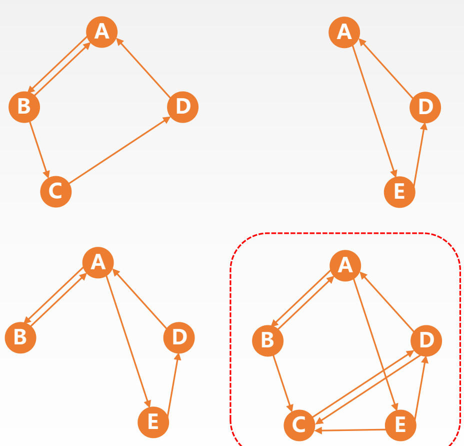


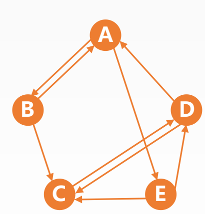
图的基本概念

有向图

强连通分量

极大强连通子图

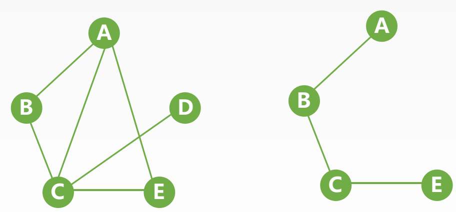




图的基本概念

生成树、生成森林

极小连通子图 连通子图且包含的边最少

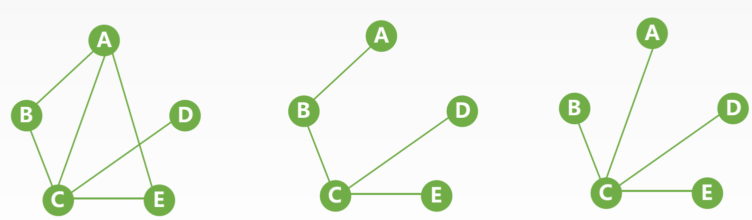


王道考研/CSKAOYAN.COM

图的基本概念

生成树、生成森林

生成树 连通图包含全部顶点的一个极小连通子图



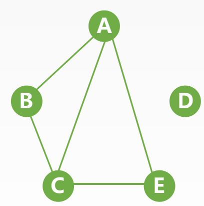
★n个顶点图的生成树有n-1条边

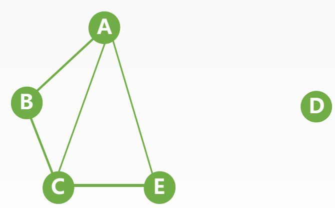
王道考研/CSKAOYAN.COM

图的基本概念

生成树、生成森林

生成森林　非连通图所有连通分量的生成树组成生成森林



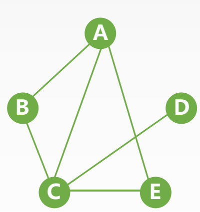


图的基本概念

顶点的度 以该顶点为一个端点的边的数目

无向图

顶点v的度为以v为端点的边的个数
记为，TD(v)。

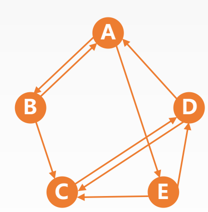


★n顶点，e条边的无向图中度的总数为

$$\sum_{i=1}^{n} TD(v_i) = 2e$$

有向图

出度：以v为起点的有向边的条数，记OD(v)
入度：以v为终点的有向边的条数，记ID(v)
顶点v的度为出度和入度之和
TD(v)=ID(v)+OD(v)

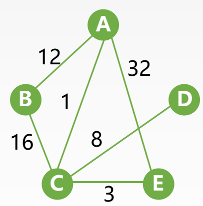


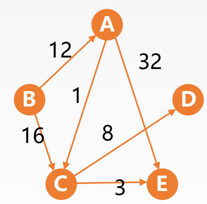
★n顶点，e条边的有向图中出度，入度为

$$\sum_{i=1}^{n} ID(v_i) = \sum_{i=1}^{n} OD(v_i) = e$$

图的基本概念

网

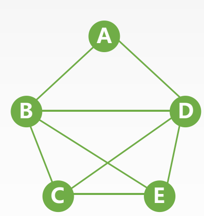




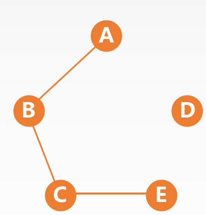
图的基本概念

稠密图&稀疏图

边多



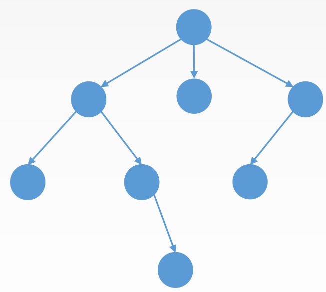
边少



稀疏稠密的界定：|E|<|V|log|V|

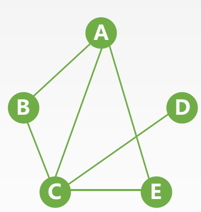
图的基本概念

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

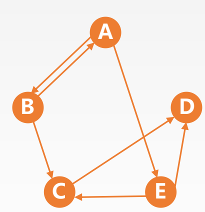


图的基本概念

路径　图中顶点v到顶点w的顶点序列，序列中顶点不重复的路径称为简单路径。



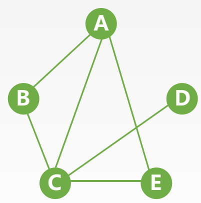
A到D的一条路径:A C D



A到D的一条路径:A E D

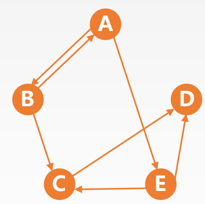
图的基本概念

路径长度 路径上边的数目，若该路径最短则称其为距离。



A到D的一条路径:A C D

路径长度 = 2

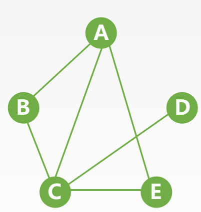


A到D的一条路径:A E D

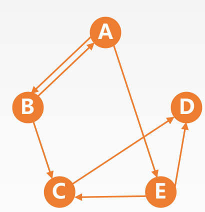
路径长度 = 2

图的基本概念

回路 第一个顶点和最后一个顶点相同的路径



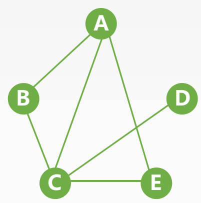
一条回路:A C E A



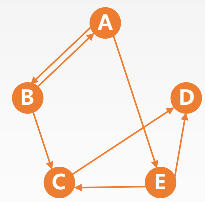
一条回路:A B A

图的基本概念

回路　第一个顶点和最后一个顶点相同的路径，若其余结点不相同则称简单回路



A到D的一条回路:A C E A



A到D的一条路径:A B A

王道考研/CSKAOYAN.COM

本节内容

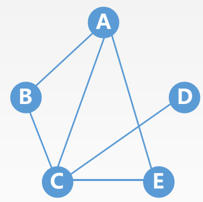
图

存储及操作

邻接矩阵法

王道考研/CSKAOYAN.COM

图的存储结构



V={A, B, C, D, E} **一维数组**

E={(A, B), (A, C), (A, E), (B, C), (C, D), (C, E)} **二维数组**

邻接矩阵

图的存储结构

邻接矩阵法

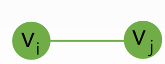
结点数为n的图G = (V, E)的邻接矩阵A是n×n的。

将G的顶点编号为 $v_1, v_2, \ldots, v_n$ （数组下标）

若< v_i, v_j >∈E，则A[i][j]=1，否则A[i][j]=0。

$$A[i][j]=\begin{cases} 1 & \text{若}(v_i, v_j)\text{或}<v_i, v_j>\text{是}E(G)\text{中的边} \\ 0 & \text{若}(v_i, v_j)\text{或}<v_i, v_j>\text{不是}E(G)\text{中的边} \end{cases}$$

A[i][j]=1



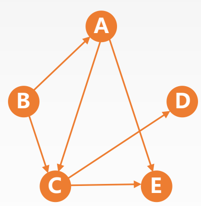
A[i][j]=1
A[j][i]=1

图的存储结构

邻接矩阵法

一维数组

0	1	2	3	4
A	B	C	D	E



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

E={<B, A>,<A, C>,<A, E>,<B, C>,<C, D>,<C, E>}

$$\text{邻接矩阵}A=\begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

二维数组

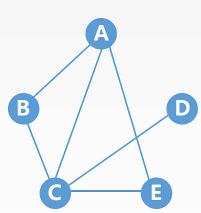
0	0	1	0	1
1	0	1	0	0
0	0	0	1	1
0	0	0	0	0
0	0	0	0	0

图的存储结构

邻接矩阵法

一维数组

0	1	2	3	4
A	B	C	D	E



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

E={(A, B), (A, C), (A, E), (B, C), (C, D), (C, E)}

$$\text{邻接矩阵}A=\begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

二维数组

0	1	1	0	1
1	0	1	0	0
1	1	0	1	1
0	0	1	0	0
0	0	1	0	0

图的存储结构

邻接矩阵法

结点数为n的图G = (V, E)的邻接矩阵A是n×n的。

将G的顶点编号为 $v_1, v_2, \ldots, v_n$（数组下标）

若（v_i, v_j）∈E，则A[i][j]=$w_{i,j}$，否则A[i][j]= ∞。

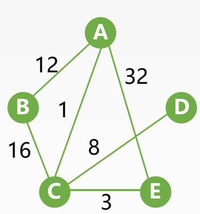
$$A[i][j]=\begin{cases} w_{i,j} & \text{若}(v_i,v_j)\text{或}<v_i,v_j>\text{是E(G)中的边} \\ \infty & \text{若}(v_i,v_j)\text{或}<v_i,v_j>\text{不是E(G)中的边} \end{cases}$$

A[i][j]=$w_{i,j}$

A[i][j]=$w_{i,j}$
A[j][i]=$w_{i,j}$

图的存储结构

邻接矩阵法



$$\text{邻接矩阵A}=\begin{pmatrix} \infty & 12 & 1 & \infty & 32 \\ 12 & \infty & 16 & \infty & \infty \\ 1 & 16 & \infty & 8 & 3 \\ \infty & \infty & 8 & \infty & \infty \\ 32 & \infty & 3 & \infty & \infty \end{pmatrix}$$

图的存储结构

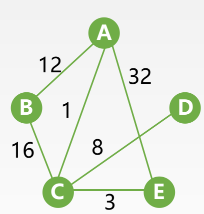
邻接矩阵法

```c
#define MaxVertexNum 100
typedef char VertexType;
typedef int EdgeType;
typedef struct{
    VertexType Vex[MaxVertexNum];
    EdgeType Edge[MaxVertexNum][MaxVertexNum];
    int vexnum,arcnum;
}MGraph;
```

★ 邻接矩阵法的空间复杂为O(n^2)

图的存储结构

性质

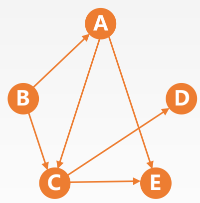


$$\begin{pmatrix} \infty & 12 & 1 & \infty & 32 \\ 12 & \infty & 16 & \infty & \infty \\ 1 & 16 & \infty & 8 & 3 \\ \infty & \infty & 8 & \infty & \infty \\ 32 & \infty & 3 & \infty & \infty \end{pmatrix}$$

- 邻接矩阵法的空间复杂为O(n^2)，适用于稠密图
- 无向图的邻接矩阵为对称矩阵
- 无向图中第i行（或第i列）非0元素（非正无穷）的个数为第i个顶点的度；
  有向图中第i行（第i列）非0元素（非正无穷）的个数为第i个顶点的出度（入度）

图的存储结构

? 设图G的邻接矩阵为A，矩阵运算 A^n 的含义



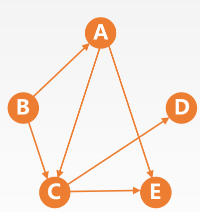
$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$A^2[2][5] = 1*1+0*0+1*1+0*0+0*0 = 2$

$A^2[2][5]=2$ 表示从顶点v2到顶点v5长度为2的路径有两条

图的存储结构

? 设图G的邻接矩阵为A，矩阵运算 A^n 的含义



$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$A^3[2][5] = 0*1+0*0+1*1+1*0+2*0 = 1$

$A^2[2][3]=1$ 表示从顶点v2到顶点v5长度为2的路径有一条

$A^3[2][5]=1$ 表示从顶点v2到顶点v5长度为3的路径有一条

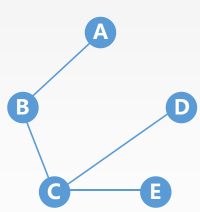
★ $A^n[i][j]$ 表示从顶点vi到顶点vj长度为n的路径有条数

本节内容

图

存储及操作
邻接表法

图的存储结构



二维数组

0	1	0	0	0
1	0	1	0	0
0	1	0	1	1
0	0	1	0	0
0	0	1	0	0

★ 邻接矩阵法存储稀疏图会有许多空间浪费

图的存储结构

邻接表法 为每一个顶点建立一个单链表存放与它相邻的边

顶点表

采用顺序存储，每个数组元素存放顶点的数据和边表的头指针

边表（出边表）

采用链式存储，单链表中存放与一个顶点相邻的所有边，一个链表结点表示一条从该顶点到链表结点顶点的边

顶点表结点

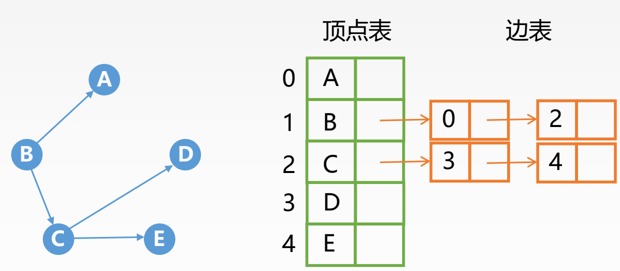
data	firstarc

边表结点

adjvex	nextarc

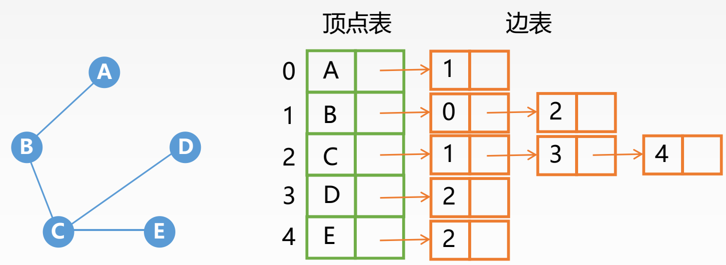
图的存储结构

邻接表法



图的存储结构

邻接表法



图的存储结构

邻接表法

边表结点

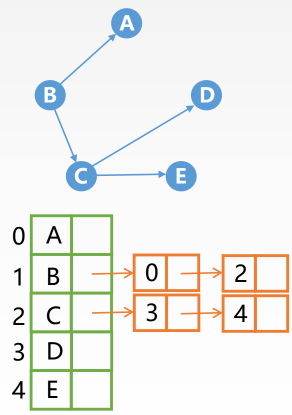
顶点表

邻接表

```
#define MaxVertexNum 100
typedef struct ArcNode{
    int adjvex;
    struct ArcNode *next;
    //InfoType info;
}ArcNode;
typedef struct VNode{
    VertexType data;
    ArcNode *first;
}VNOode,AdjList[MaxVertexNum];
typedef struct{
    AdjList vetices;
    int vexnum, arcnum;
}ALGraph;
```

图的存储结构

特点



- 若G为无向图，存储空间为O（|V|+2|E|）
  若G为有向图，存储空间为O（|V|+|E|）
- 邻接表更加适用于稀疏图
- 若G为无向图，则结点的度为该结点边表的长度
  若G为有向图，则结点的出度为该结点边表的长度，计算入度则要遍历整个邻接表
- 邻接表不唯一，边表结点的顺序根据算法和输入的不同可能会不同

图的存储结构

0	1	0	0	0
1	0	1	0	0
0	1	0	1	1
0	0	1	0	0
0	0	1	0	0

邻接矩阵 VS 邻接表

适用稠密图	适用性	适用稀疏图
顺序存储	存储方式	顺序存储+链式存储
效率高	判断两顶点间是否存在边	效率低
效率低	找出某顶点相邻的边	效率高

本节内容

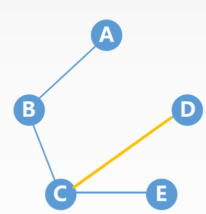
图

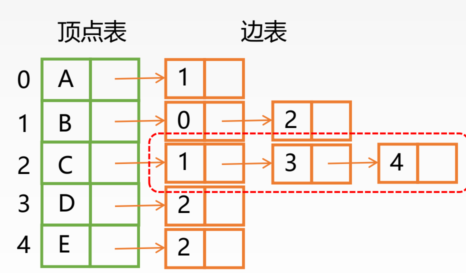
存储及操作
基本操作

图的基本操作

Adjacent(G, x, y)　判断图G是否存在边<x, y>或（x, y）

0	1	2	3	4
A	B	C	D	E
0	1	0	0	0
1	0	1	0	0
0	1	0	1	1
0	0	1	0	0
0	0	1	0	0



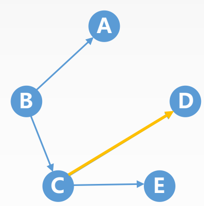


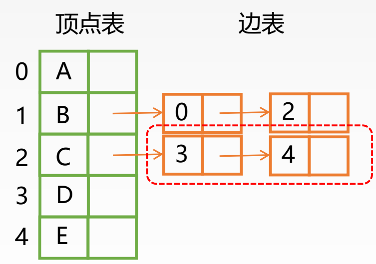
图的基本操作

Adjacent(G, x, y)　判断图G是否存在边<x, y>或（x, y）

0	1	2	3	4
A	B	C	D	E

0	0	0	0	0
1	0	1	0	0
0	0	0	1	1
0	0	0	0	0
0	0	0	0	0



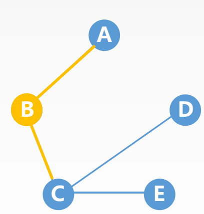


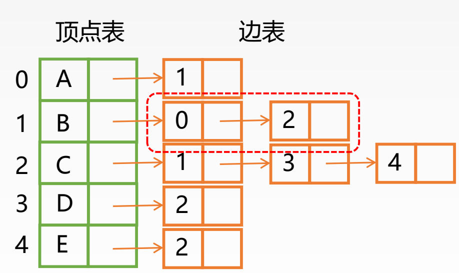
图的基本操作

Neighbors(G, x)　列出图G中与结点x邻接的边

0	1	2	3	4
A	B	C	D	E

0	1	0	0	0
1	0	1	0	0
0	1	0	1	1
0	0	1	0	0
0	0	1	0	0

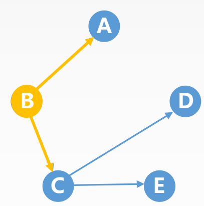


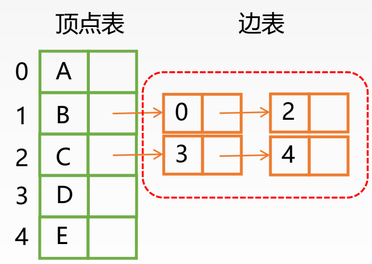


图的基本操作

Neighbors(G, x) 列出图G中与结点x邻接的边

0	1	2	3	4
A	B	C	D	E
0	0	0	0	0
1	0	1	0	0
0	0	0	1	1
0	0	0	0	0
0	0	0	0	0

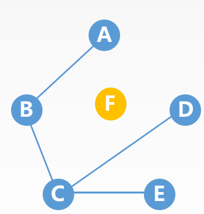


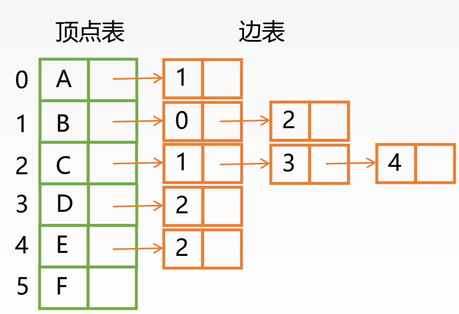


图的基本操作

InsertVertex(G, x) 在图G中插入顶点x

00	11	22	33	44	5
A	B	C	D	E	F
0	1	0	0	0	0
1	0	1	0	0	0
0	1	0	1	1	0
0	0	1	0	0	0
0	0	1	0	0	0
0	0	0	0	0	0



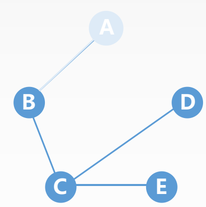


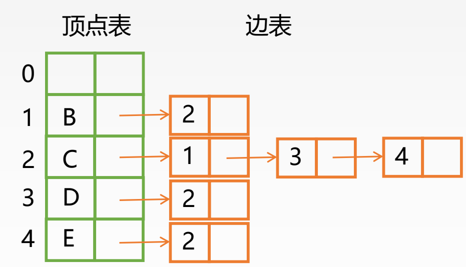
图的基本操作

DeleteVertex(G, x)　从图G中删除顶点x

0	1	2	3	4
	B	C	D	E

	0	1	0	0
	1	0	1	1
	0	1	0	0
	0	1	0	0



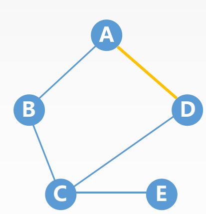


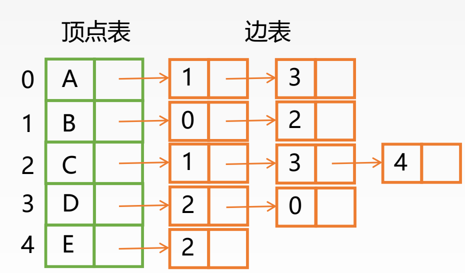
图的基本操作

AddEdge(G, x, y)　若无向边（x, y）或者有向边<x, y>不存在，则向图G中添加该边

0	1	2	3	4
A	B	C	D	E

0	1	0	0	0
1	0	1	0	0
0	1	0	1	1
0	0	1	0	0
0	0	1	0	0

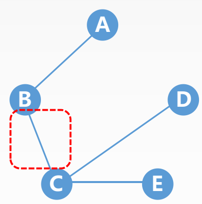


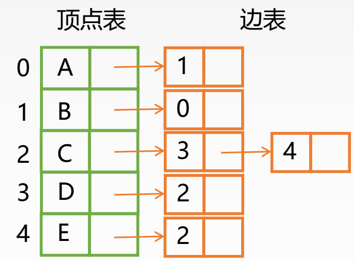


图的基本操作

RemoveEdge(G, x, y) 若无向边（x, y）或者有向边<x, y>存在，则在图G中删除该边

0	1	2	3	4
A	B	C	D	E
0	1	0	0	0
1	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	0	0





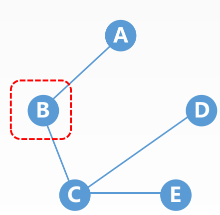
王道考研/CSKAOYAN.COM

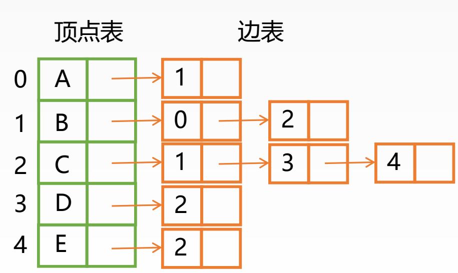
图的基本操作

FirstNeighbor(G, x) 求图G中顶点X的第一个邻接点，若有则返回顶点号。若没有邻接点或图不存在x，则返回-1。

NextNeighbor(G, x) 假设图G中顶点y是顶点x的一个邻接点，返回除y之外顶点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。

0	1	2	3	4
A	B	C	D	E
0	1	0	0	0
1	0	1	0	0
0	1	0	1	1
0	0	1	0	0
0	0	1	0	0





王道考研/CSKAOYAN.COM

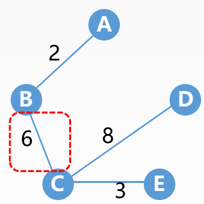
图的基本操作

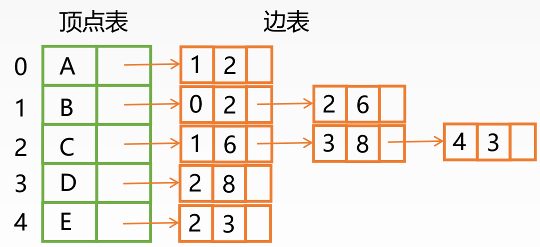
Get_edge_value(G, x, y) 获取图G中边(x, y)或<x, y>对应的权值v。

Set_edge_value(G, x, y) 设置图G中边(x, y)或<x,y>对应的权值为v。

0	1	2	3	4
A	B	C	D	E

0	2	0	0	0
2	0	6	0	0
0	6	0	8	3
0	0	8	0	0
0	0	3	0	0





本节内容

图

存储及操作

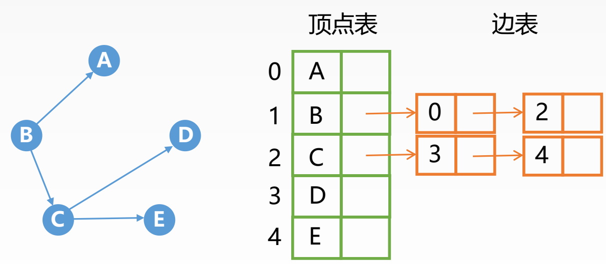
十字链表

十字链表

十字链表 有向图的一种链式存储结构

顶点表结点	
data	firstarc

边表结点	
adjvex	nextarc

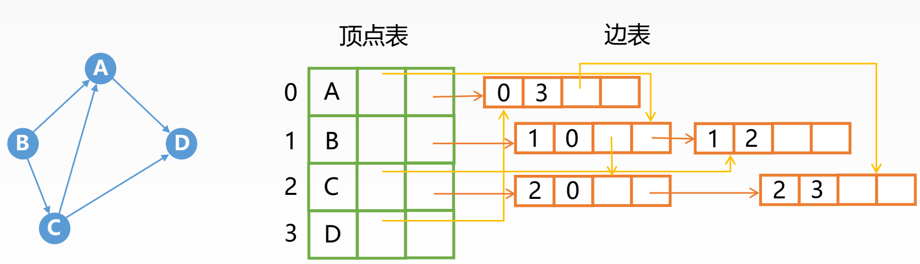


十字链表

十字链表 有向图的一种链式存储结构

顶点表结点		
data	firstin	firstout

边表结点				
tailvex	headvex	hlink	tlink	info



十字链表

十字链表

```
#define MaxVertexNum 100
typedef struct ArcNode{
    int tailvex, headvex;
    struct ArcNode *hlink, *tlink;
    //InfoType info;
}ArcNode;
typedef struct VNode{
    VertexType data;
    ArcNode *firstin, *firstout;
}VNode;
typedef struct{
    VNode xlist[MaxVertexNum];
    int vexnum, arcnum;
}GLGraph;
```


本节内容

图

存储及操作

邻接多重表

邻接多重表

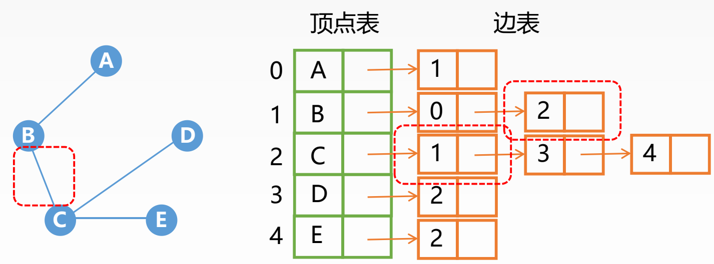
邻接多重表 无向图的一种链式存储结构

顶点表结点

data	firstarc

边表结点

adjvex	nextarc



邻接多重表

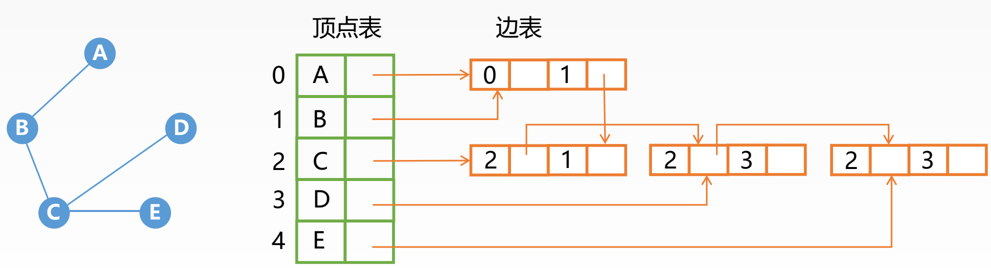
邻接多重表 无向图的一种链式存储结构

顶点表结点

data	firstedge

边表结点

ivex	ilink	jvex	jlink	info	mark



邻接多重表

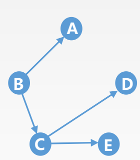
邻接多重表

```
#define MaxVertexNum 100
typedef struct ArcNode{
    int ivex, jvex;
    struct ArcNode *ilink, *jlink;
    //InfoType info;
    //bool mark;
}ArcNode;
typedef struct VNode{
    VertexType data;
    ArcNode *firstedge;
}VNode;
typedef struct{
    VNode adjmulist[MaxVertexNum];
    int vexnum, arcnum;
}AMLGraph;
```


总结

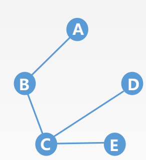
十字链表 & 邻接多重表

链式存储结构



有向图

无向图



顶点表结点

data	firstin	firstout

顶点表结点

data	firstedge

边表结点

tailvex	headvex	hlink	tlink	info

边表结点

ivex	ilink	jvex	jlink	info	mark

本节内容

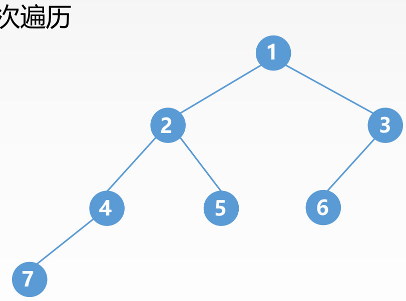
图

图的遍历
广度优先搜索

广度优先搜索

图的遍历 从图中某一顶点出发，按照某种搜索方法沿着图中的边对图中的所有顶点访问一次且仅访问一次。

树的层次遍历

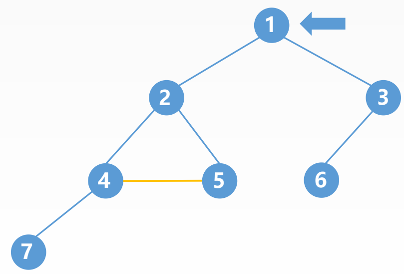


广度优先搜索

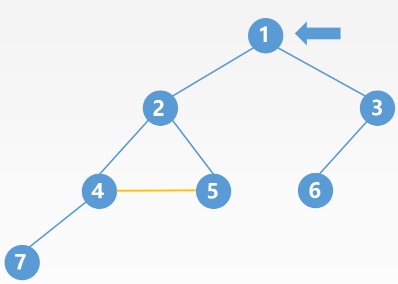
广度优先搜索

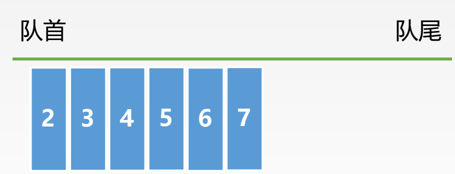
- 首先访问起始顶点v;
- 接着由出发依次访问v的各个未被访问过的邻接顶点 $w_1,w_2,\ldots,w_i$;
- 然后依次访问$w_1,w_2,\ldots,w_i$的所有未被访问过的邻接顶点;
- 在从这些访问过的顶点出发，访问它们所有未被访问过的邻接顶点;
- ……，以此类推;

队列 + 辅助标记数组



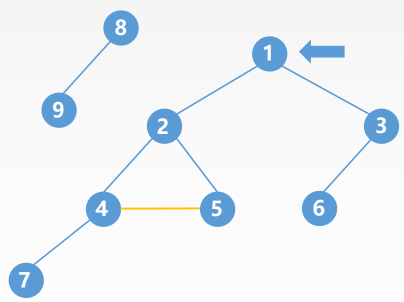
广度优先搜索





	0	1	2	3	4	5	6
visit	0	0	0	0	0	0	0

广度优先搜索




```
bool visited[MAX_TREE_SIZE];
void BFSTraverse(Graph G){
    for(int i=0;i<G.vexnum;++i)
        visited[i]=FALSE;
    InitQueue(Q);
    for(int i;i<G.vexnum;++i)
        if(!visited[i])
            BFS(G, i);
}
void BFS(Graph G, int v){
    visit(v);
    visited[v] = TRUE;
    EnQueue(Q, v);
    while(!isEmpty(Q)){
        DeQueue(Q, v);
        for(w=FirstNeighbor(G, v);w>=0;w=NextNeighbor(G, v, w))
            if(!visited[w]){
                visit[w];
                visited[w]=TRUE;
                EnQueue(Q, w);
            }
    }
}
```


广度优先搜索

BFS算法的性能分析

空间复杂度　　O(|V|)

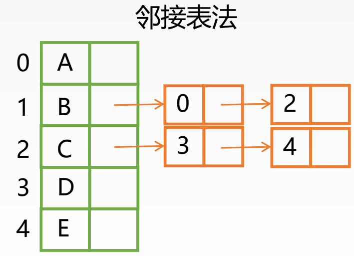
时间复杂度

邻接矩阵法

0	0	0	0	0
1	0	1	0	0
0	0	0	1	1
0	0	0	0	0
0	0	0	0	0

$O(|V|^2)$

邻接表法

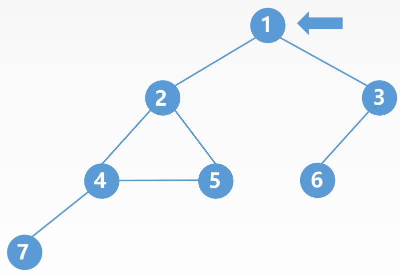


$O(|V|+|E|)$

广度优先搜索

无权图单源最短路径问题

定义从顶点u到顶点v的最短路径d(u,v)为从u到v的任何路径中最少的边数；
若从u到v没有通路，则d(u,v)=∞。



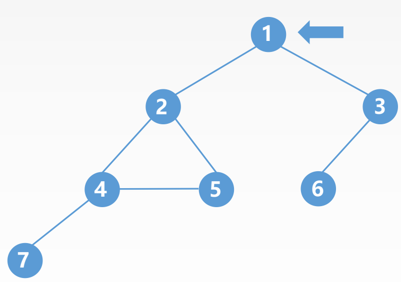

```
void BFS_MIN_Distance(Graph G, int u){
    for(int i=0; i<G.vexnum; ++i)
        d[i] = MAX;
    visited[u] = TRUE;
    d[u] = 0;
    EnQueue(Q, u);
    while(!isEmpty(Q)){
        DeQueue(Q, u);
        for(w=FirstNeighbor(G, u); w>=0; w=NextNeighbor(G, u, w))
            if(!visit[w]){
                visited[w] = TRUE;
                d[w] = d[u]+1;
                EnQueue(Q, w);
            }
    }
}
```

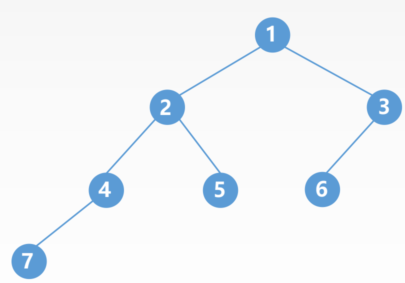
王道考研/CSKAOYAN.COM

广度优先搜索

广度优先生成树

在广度遍历过程中，我们可以得到一棵遍历树，称为广度优先生成树（生成森林）。





★ 邻接矩阵法的广度优先生成树唯一，邻接表法的不唯一

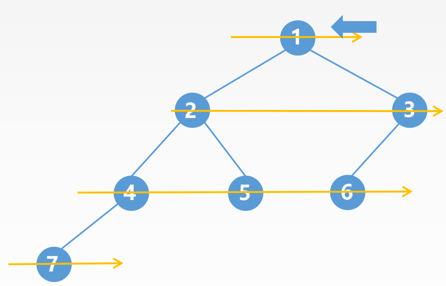
王道考研/CSKAOYAN.COM

本节内容

图

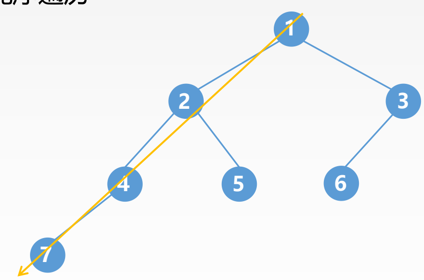
图的遍历
深度优先搜索

深度优先搜索



深度优先搜索

树的先序遍历

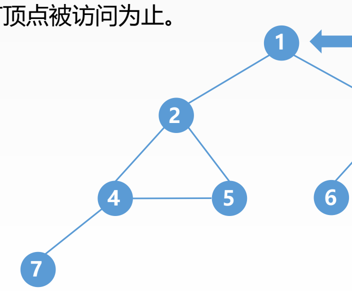


深度优先搜索

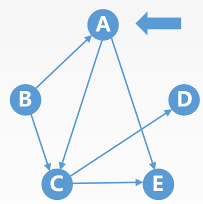
深度优先搜索 DFS

- 首先访问起始顶点v;
- 接着由v出发访问v的任意一个邻接且未被访问的邻接顶点 w_i；
- 然后再访问与w_i邻接且未被访问的任意顶点y_i；
- 若w_i没有邻接且未被访问的顶点时，退回到它的上一层顶点v；
- 重复上述过程，直到所有顶点被访问为止。

递归 + 辅助标记数组
(栈)



深度优先搜索



DFS序列：ACDEB

```
bool visited[MAX_TREE_SIZE];
void DFSTraverse(Graph G){
    for(int i=0;i<G.vexnum;++i)
        visited[i]=FALSE;
    for(int i=0;i<G.vexnum;++i)
        if(!visited[i])
            DFS(G, i);
}
void DFS(Graph G, int v){
    visit(v);
    visited[v]=TRUE;
    for(w=FirstNeighbor(G, v); w>=0; w=NextNeighbor(G, v, w))
        if(!visited[w])
            DFS(G, w);
}
```

★ 邻接矩阵法的DFS（BFS）序列唯一，邻接表法的不唯一

深度优先搜索

DFS算法的性能分析

空间复杂度　O(|V|)　工作栈

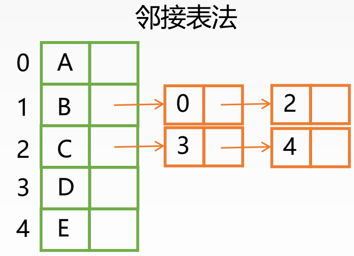
时间复杂度

邻接矩阵法

0	0	0	0	0
1	0	1	0	0
0	0	0	1	1
0	0	0	0	0
0	0	0	0	0

$O(|V|^2)$

邻接表法

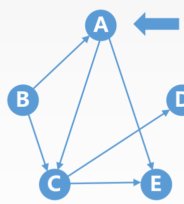


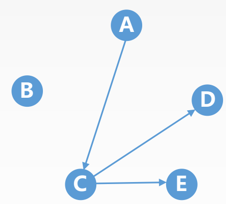
$O(|V|+|E|)$

深度优先搜索

深度优先生成树

在深度遍历过程中，我们可以得到一棵遍历树，称为深度优先生成树（生成森林）。

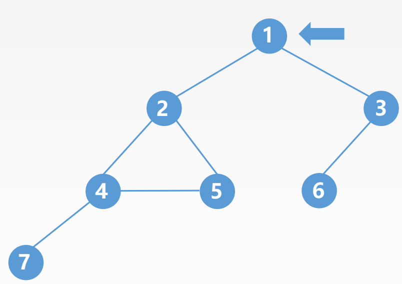




★ 邻接矩阵法的深度优先生成树唯一，邻接表法的不唯一

深度优先搜索

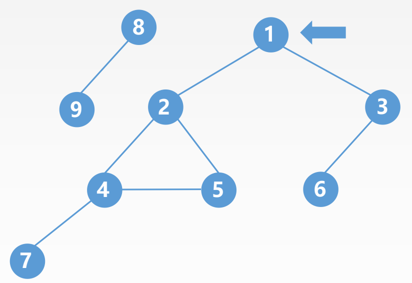
遍历与连通性问题



★ 在无向图当中，在任意结点出发进行一次遍历（调用一次BFS或DFS），若能访问全部结点，说明该无向图是连通的。

深度优先搜索

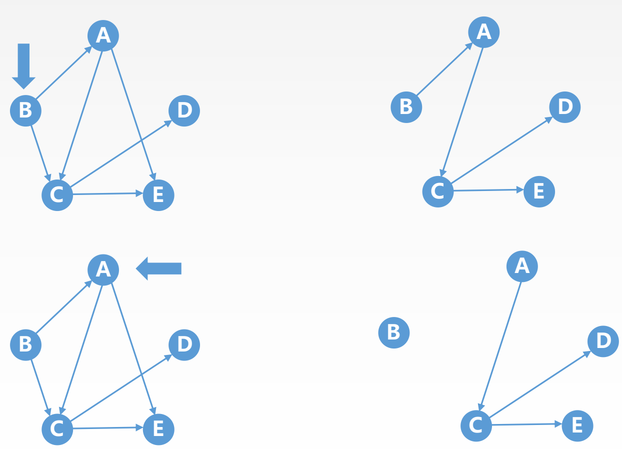
遍历与连通性问题



★在无向图中，调用遍历函数（BFS或DFS）的次数为连通分量的个数。

深度优先搜索

遍历与连通性问题



总结

本节内容

图

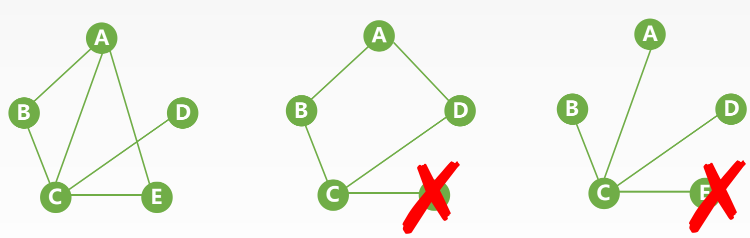
图的应用
最小生成树

最小生成树

最小生成树

最小生成树

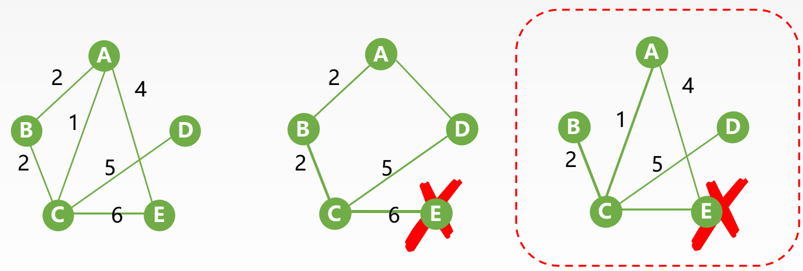
生成树 连通图包含全部顶点的一个极小连通子图



最小生成树

最小生成树

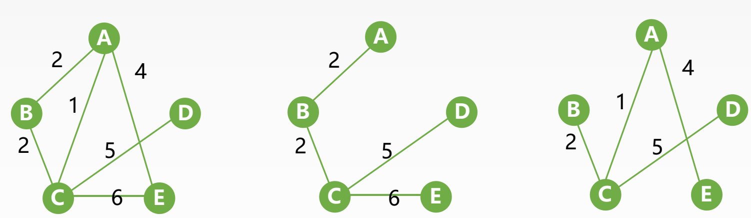
生成树 连通图包含全部顶点的一个极小连通子图



最小生成树

最小生成树

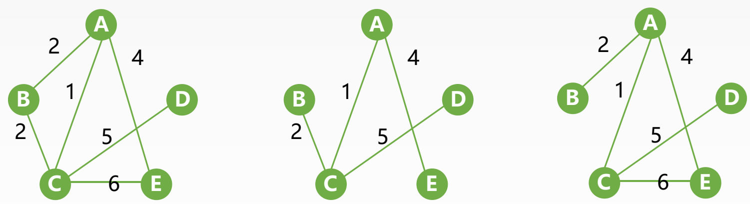
对于带权无向连通图G=（V，E），G的所有生成树当中边的权值之和最小的生成树为G的最小生成树（MST）。



最小生成树

性质

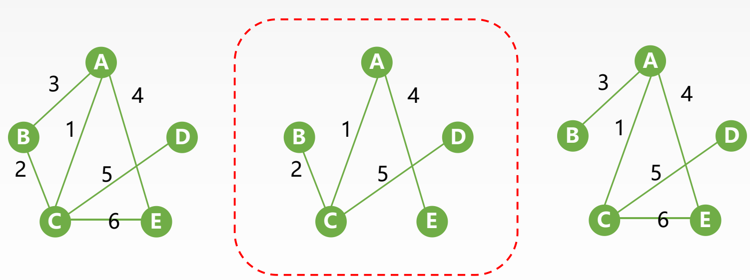
不一定唯一



最小生成树

性质

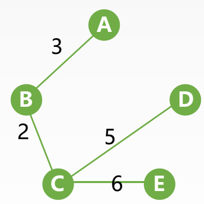
不一定唯一



最小生成树

性质

不一定唯一



最小生成树

性质

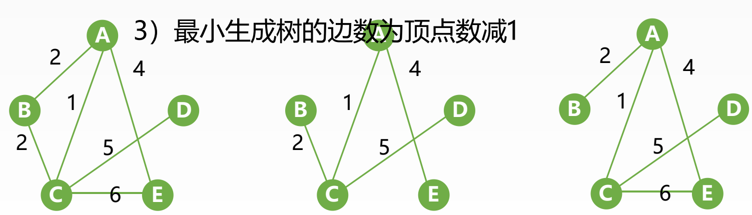
不一定唯一

1）最小生成树不一定唯一，即最小生成树的树形不一定唯一。当带权无向连通图G的各边权值不等时或G只有结点数减1条边时，MST唯一。

唯一

2）最小生成树的权值是唯一的，且是最小。

3）最小生成树的边数为顶点数减1



最小生成树

算法

```
GENRIC_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);
}
```

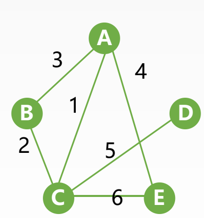
Prim　　　Kruskal

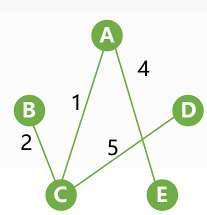
最小生成树

Prim

初始化：向空的结果树$T=(V_T, E_T)$中添加图$G=(V, E)$的任一顶点u_0，使$V_T=\{u_0\}$，E_T为空集；

循环(直到$V_T=V$)：从图G中选择满足$\{(u, v) | u\in V_T, v\in V-V_T\}$且具有最小权值的边(u, v)，并置$V_T=V_T\cup\{v\}$，$E_T=E_T\cup\{(u, v)\}$。





最小生成树

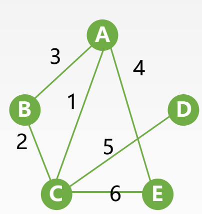
Prim

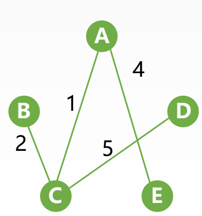
```
void Prim(G, T){
    T = 空集;
    U = {w};
    while((V-U)!=空集){
        设(u, v)是使 u∈U与v∈(V-U),且权值最小的边;
        T = T∪{(u, v)};
        U = U∪{v};
    }
}
```

辅助数组:
min_weight[n]　　　adjvex[n]

最小生成树

Prim





min_weight	0	3	1	∞	4
adjvex	0	0	0	0	0

0	2	0	5	4
0	0	0	0	0

0	0	0	5	4
0	2	0	0	0

0	0	0	5	0
0	2	0	0	0

0	0	0	0	0
0	2	0	2	0

最小生成树

```
void MST_Prim(Graph G){
    int min_weight[G.vexnum];
    int adjvex[G.vexnum];
    for(int i=0; i<G.vexnum; i++){
        min_weight[i] = G.Edge[0][i];
        adjvex[i] = 0;
    }

    int min_arc;
    int min_vex;
    for(int i=1; i<G.vexnum; i++){
        min_arc = MAX;
        for(int j=1; j<G.vexnum; j++)
            if(min_weight[j]!=0 && min_weight[j]<min_arc){
                min_arc = min_weight[j];
                min_vex = j;
            }
        min_weight[min_vex] = 0;
        for(int j=0; j<G.vexnum; j++){
            if(min_weight[j]!=0 && G.Edge[min_arc][j]<min_weight[j]){
                min_weight[j] = G.Edge[min_arc][j];
                adjvex[j] = min_arc;
            }
        }
    }
}
```

$O(|V|^2)$

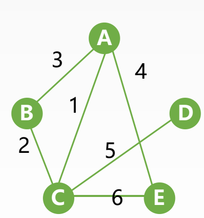
★ 适于稠密图

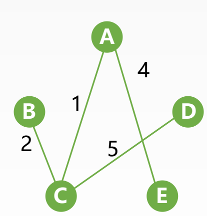
最小生成树

Kruskal

初始化：$V_T = V$，E_T = 空集 。即是每个顶点构成一棵独立的树，T是一个仅含|V|个顶点的森林；

循环(直到T为树)：按图G的边的权值递增的顺序依次从 $E - E_T$ 中选择一条边，若这条边加入后不构成回路，则将其加入 E_T，否则舍弃。

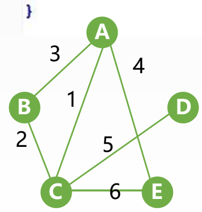


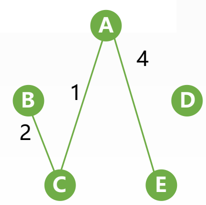


最小生成树

Kruskal

```
void Kruskal(V, T){
    T=V;
    numS=n;
    while(numS>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)};
            numS--;
        }
    }
}
```





最小生成树

Kruskal

```
void Kruskal(V, T){
    T=V;
    numS=n;
    while(numS>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)};
            numS--;
        }
    }
}
```

堆排序Sort()　　　　并查集

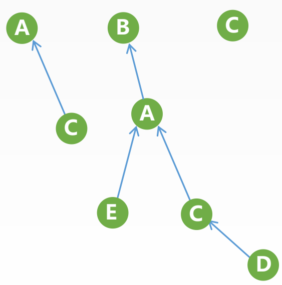
最小生成树

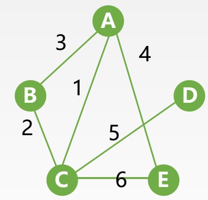
Kruskal

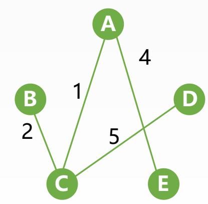
堆排序　AC　BC　AB　AE　CD　CE

并查集　parent

1	-1	0	2	0







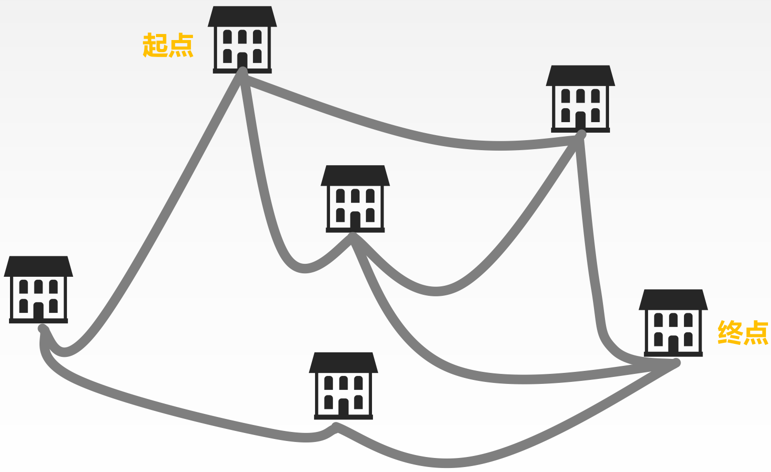
本节内容

图

图的应用

最短路径

最短路径



最短路径

最短路径 两个顶点之间带权路径长度最短的路径为最短路径。

在带权图当中，把从一个顶点v到另一个顶点u所经历的边的权值之和称为，路径的带权路径长度。

最短路径

Dijkstra 带权图单源最短路径

辅助数组

s[]：标记已经计算完成的顶点。
数组中的值全部初始化为0。源点下标的值初始化为1。

dist[]：记录从源点v0到其他各顶点当前的最短路径长度。
数组中的值初始化为源点到各个顶点边的权值，即dist[i]=arcs[0][i];。

path[]：记录从最短路径中顶点的前驱顶点，即path[i]为v到vi最短路径上vi的前驱顶点。
数组中的值初始化：
若源点v0到该顶点vi有一条有向边（无向边），则另path[i]=0;
否则path[i]=-1;

王道考研/CSKAOYAN.COM

最短路径

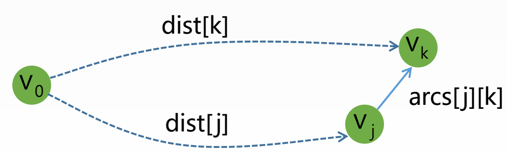
Dijkstra 带权图单源最短路径

1) 初始化数组，并集合S初始为{0};

2) 从顶点集合V-S中选出v_j，满足dist[j]=Min{dist[i]| $v_i \in$V-S},
v_j 就是当前求得的最短路径的终点，并另S∪{j};

3) 修改此时从v_0出发到集合V-S上任一顶点v_k最短路径的长度：
若dist[j]+arcs[j][k]<dist[k]
则令dis[k]=dist[j]+arcs[j][k];　　path[k]=j;

4) 重复2）、3）操作n-1次，直到S中包含全部顶点；



王道考研/CSKAOYAN.COM

最短路径

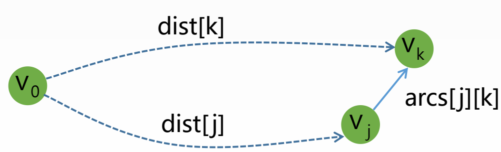
Dijkstra 带权图单源最短路径

1) 初始化数组，并集合S初始为{0};

2) 从顶点集合V-S中选出v_j，满足dist[j]=Min{dist[i]| v_j∈V-S},
v_j 就是当前求得的最短路径的终点，并另S∪{j};

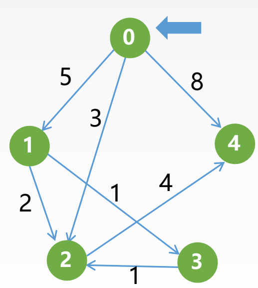
3) 修改此时从 v_0 出发到集合V-S上任一顶点 v_k 最短路径的长度:
若dist[j]+arcs[j][k]<dist[k]
则令dis[k]=dist[j]+arcs[j][k];　　path[k]=j;

4) 重复2）、3）操作n-1次，直到S中包含全部顶点;



最短路径

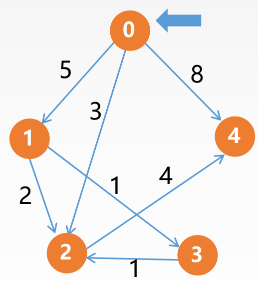
Dijkstra



	dist[0]	dist[1]	dist[2]	dist[3]	dist[4]	s[0]	s[1]	s[2]	s[3]	s[4]	path[0]	path[1]	path[2]	path[3]	path[4]
初始化	0	5	3	∞	8	1	0	0	0	0	-1	0	0	-1	0
第一轮	0	5	3	∞	7	1	0	1	0	0	-1	0	0	-1	2
第二轮	0	5	3	6	7	1	1	1	0	0	-1	0	0	1	2
第三轮	0	5	3	6	7	1	1	1	1	0	-1	0	0	1	2
第四轮	0	5	3	6	7	1	1	1	1	1	-1	0	0	1	2

最短路径

Dijkstra



	dist[]					s[]					path[]				
	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
第四轮	0	5	3	6	7	1	1	1	1	1	-1	0	0	1	2

路径	长度	序列
0 -> 1	5	0 → 1
0 -> 2	3	0 → 2
0 -> 3	6	0 → 1 → 3
0 -> 4	7	0 → 2 → 4

最短路径

```
void Dijkstra(Graph G, int v){
    int s[G.vexnum];
    int path[G.vexnum];
    int dist[G.vexnum];
    for(int i=0; i<G.vexnum; i++){
        dist[i]=G.edge[v][i];
        s[i]=0;
        if(G.edge[v][i] < MAX)
            path[i]=v;
        else
            path[i]=-1;
    }
    s[v]=1;
    path[v]=-1;
```

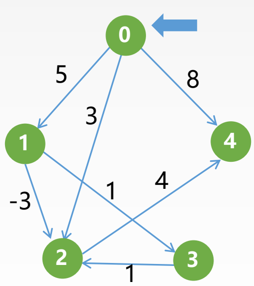
最短路径

```
for(i=0; i<G.vexnum; i++){
    int min = MAX;
    int u;
    for(int j=0; j<G.vexnum; j++)
        if(s[j]==0 && dist[j]<min){
            min=dist[j];
            u=j;
        }
    s[u]=1;

    for(int j=0; j<G.vexnum; j++)
        if(s[j]==0 && dist[u]+G.Edge[u][j]<dist[j]){
            dist[j]=dist[u]+G.Edges[u][i];
            path[j]=u;
        }
    }
}
```

$O(|V|^2)$

最短路径



	dist[] 0	1	2	3	4	s[] 0	1	2	3	4	path[] 0	1	2	3	4
初始化	0	5	3	∞	8	1	0	0	0	0	-1	0	0	-1	0
第一轮	0	5	3	∞	7	1	0	1	0	0	-1	0	0	-1	2
第二轮	0	5	3	6	7										

3) 修改此时从 v_0 出发到集合V-S上任一顶点 v_k 最短路径的长度：
若dist[j]+arcs[j][k]<dist[k]
则令dis[k]=dist[j]+arcs[j][k];　　path[k]=j;

★ Dijkstra算法并不适用于含有负权边的图

最短路径

Floyd

各顶点之间的最短路径

算法思想

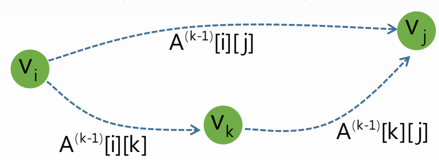
递推产生一个n阶方阵序列 $A^{(-1)}$, $A^{(0)}$,…, $A^{(k)}$,…, $A^{(n-1)}$

$A^{(k)}[i][j]$：顶点v_i到v_j的最短路径长度，且该路径经过的顶点编号不大于k

递推公式

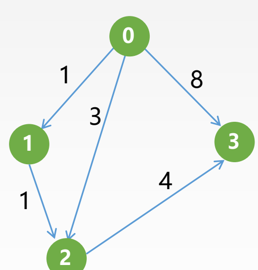
初始化：$A^{(-1)}[i][j] = arcs[i][j]$

递推方法：$A^{(k)}[i][j] = Min\{A^{(k-1)}[i][j],\ A^{(k-1)}[i][k] + A^{(k-1)}[k][j]\}, k = 0,1,\ldots,n-1$



最短路径

Floyd



初始化 $A^{(-1)} = \begin{bmatrix} 0 & 1 & 3 & 8 \\ \infty & 0 & 1 & \infty \\ \infty & \infty & 0 & 4 \\ \infty & \infty & \infty & 0 \end{bmatrix}$

$A^{(0)} = \begin{bmatrix} 0 & 1 & 3 & 8 \\ \infty & 0 & 1 & \infty \\ \infty & \infty & 0 & 4 \\ \infty & \infty & \infty & 0 \end{bmatrix}$　$A^{(1)} = \begin{bmatrix} 0 & 1 & 2 & 8 \\ \infty & 0 & 1 & \infty \\ \infty & \infty & 0 & 4 \\ \infty & \infty & \infty & 0 \end{bmatrix}$

$A^{(2)} = \begin{bmatrix} 0 & 1 & 2 & 6 \\ \infty & 0 & 1 & 5 \\ \infty & \infty & 0 & 4 \\ \infty & \infty & \infty & 0 \end{bmatrix}$　$A^{(3)} = \begin{bmatrix} 0 & 1 & 2 & 6 \\ \infty & 0 & 1 & 5 \\ \infty & \infty & 0 & 4 \\ \infty & \infty & \infty & 0 \end{bmatrix}$

最短路径

Floyd

```
void Floyd(Graph G){
    int A[G.vexnum][G.vexnum];
    for(int i=0; i<G.vexnum; i++)
        for(int j=0; j<G.vexnum; j++)
            A[i][j]=G.Edge[i][j];
    for(int k=0; k<G.vexnum; k++)
        for(int i=0; i<G.vexnum; i++)
            for(int j=0; j<G.vexnum; j++)
                if(A[i][j] > A[i][k]+A[k][j])
                    A[i][j] = A[i][k]+A[k][j];
}
```

$O(|V|^3)$

本节内容

图

图的应用

拓扑排序

拓扑排序

有向无环图 不存在环的有向图，简称DAG图。

AOV网 若用一个DAG图表示一个工程，其顶点表示活动，用有向边<vi,vj>表示活动vi先于活动vj进行的传递关系，则将这种DAG称为顶点表示活动网络，记为AOV网。

拓扑排序 对DAG所有顶点的一种排序，使若存在一条从顶点A到顶点B的路径，在排序中B排在A的后面。

拓扑排序

拓扑排序

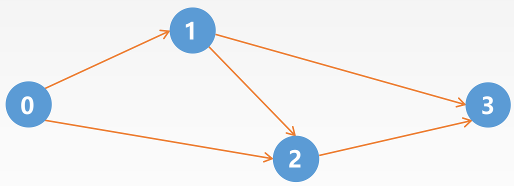
算法思想

1）从DAG图中选择一个没有前驱的顶点并输出

2）从图中删除该顶点和所有以它为起点的有向边

3）重复1）、2）直到当前的DAG图为空或当前图中不存在无前驱的顶点为止。后一种情况说明图中有环。

拓扑排序

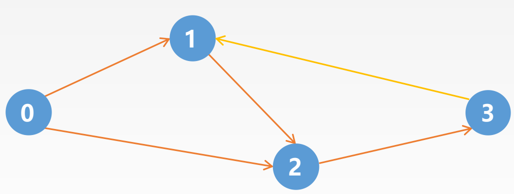


	0	1	2	3
初始化	0	1	2	2
	-	0	1	2
	-	-	0	1
	-	-	-	0

拓扑排序的序列：{0，1，2，3}

王道考研/CSKAOYAN.COM

拓扑排序

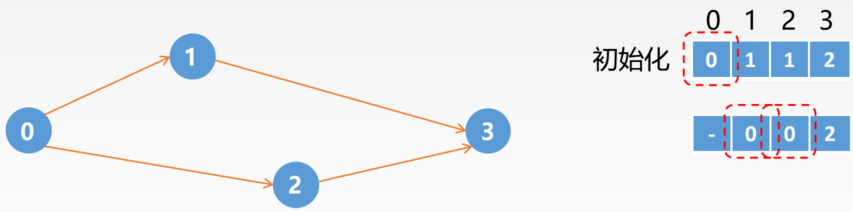


	0	1	2	3
初始化	0	2	2	1
	-	1	1	1

★ 算法结束时没有访问所有顶点，则存在以剩下顶点组成的环。

王道考研/CSKAOYAN.COM

拓扑排序



拓扑排序的序列： {0，1，2，3}
或 {0，2，1，3}

★ 拓扑排序的结果不一定唯一

拓扑排序

```
bool TopologicalSort(Graph G){
    InitStack(S);
    for(int i=0; i<G.vexnum; i++)
        if(indegree[i]==0)
            Push(S, i);
    int count=0;
    while(!isEmpty(S)){
        Pop(S, i);
        print[count++]=i;
        for(p=G.Vertices[i].firstarc; p; p=p->nextarc){
            v = p->adjvex;
            if(!(--indegree[v]))
                Push(S, v);
        }
    }
    if(count < G.vexnum)
        return false;
    else
        return true;
}
```

$O(|V|+|E|)$

拓扑排序

拓扑排序

若邻接矩阵为三角矩阵，则存在拓扑排序；反之不一定成立。

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad v_i \to v_j \qquad v_i < v_j$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad v_i \to v_j \qquad v_i > v_j$$

本节内容

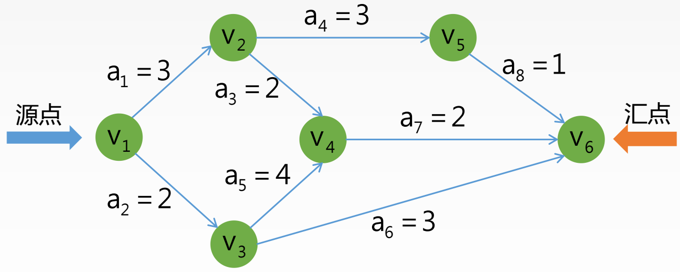
图

图的应用

关键路径

关键路径

AOE网 在有向带权图中，以顶点表示事件，以有向边表示活动，以边上权值表示完成该活动的开销（如完成活动所需要的时间），则称这种有向图为用边表示活动的网络，简称AOE网。



关键路径 从原点到汇点最大路径长度的路径称为关键路径，关键路径上的活动为关键活动。

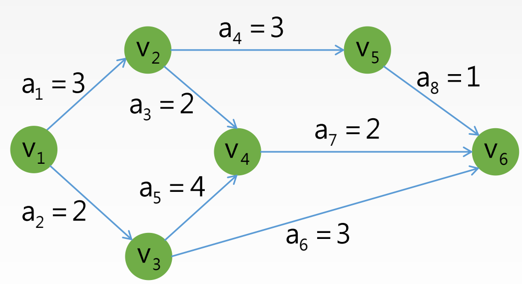
关键路径

1、事件 v_k 的最早发生时间 $v_e(k)$

$v_e(\text{源点})=0$

$v_e(k)=\text{Max}\{v_e(j)+\text{Weight}（v_j, v_k)\}$

	1	2	3	4	5	6
$v_e(i)$	0	3	2	6	6	8



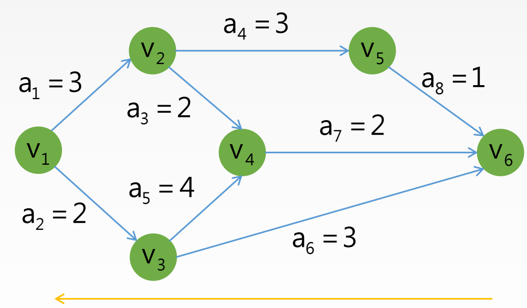
关键路径

2、事件 v_k 的最迟发生时间 $v_l(k)$

$v_l(汇点)=v_e(汇点)$

$v_l(j)=Min\{v_e(k)-Weight(v_j, v_k)\}$

	1	2	3	4	5	6
$v_l(i)$	0	4	2	6	7	8



关键路径

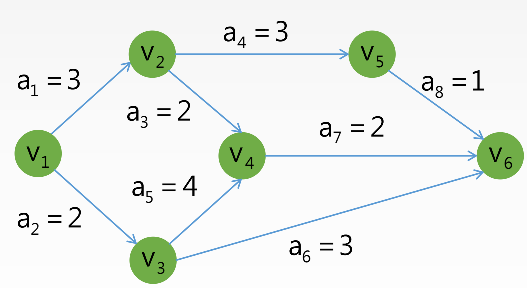
3、活动 a_i 的最早开始时间 e(i)

若存在 $<v_k, v_j>$ 表示活动 a_i，

则 $e(i)=v_e(k)$

	1	2	3	4	5	6
$v_e(i)$	0	3	2	6	6	8

	1	2	3	4	5	6	7	8
e(i)	0	0	3	3	2	2	6	6



关键路径

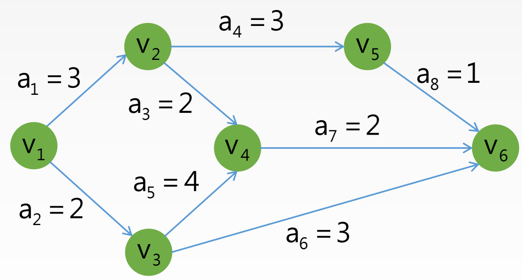
4、活动 a_i 的最迟开始时间 $l(i)$

若存在 $<v_k, v_j>$ 表示活动 a_i,

则 $l(i) = v_l(j) - Weight(v_k, v_j)$

	1	2	3	4	5	6
$v_l(i)$	0	4	2	6	7	8

	1	2	3	4	5	6	7	8
$l(i)$	1	0	4	4	2	5	6	7

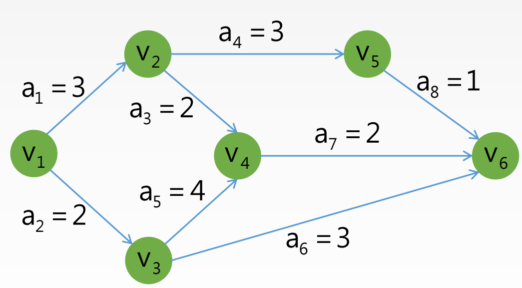


关键路径

5、活动 a_i 的差额 $d(i) = l(i) - e(i)$

	1	2	3	4	5	6	7	8
$e(i)$	0	0	3	3	2	2	6	6
$l(i)$	1	0	4	4	2	5	6	7
$d(i)$	1	0	1	1	0	3	0	1

关键路径：{ a_2, a_5, a_7 }



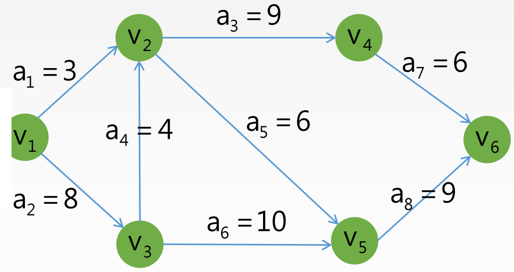
★缩短关键活动时间可以加快整个工程，但缩短到一定大小时关键路径会发生改变。

关键路径

	1	2	3	4	5	6
$v_e(i)$	0	12	8	21	18	27
$v_l(i)$	0	12	8	21	18	27

★ 当网中关键路径不唯一时，只有加快的关键活动或关键活动组合包括在所有的关键路径上才能缩短工期。

l(i)	9	0	12	8	12	8	21	18
d(i)	9	0	0	0	0	0	0	0



关键路径：{ a_2, a_4, a_3, a_7 } { a_2, a_4, a_5, a_8 } { a_2, a_6, a_8 }