jyang
@cs.ecnu.edu.cn

第七章　图

1 图的基本概念

2 图的存储结构

3 图的遍历与求图的连通分量

4 生成树和最小代价生成树

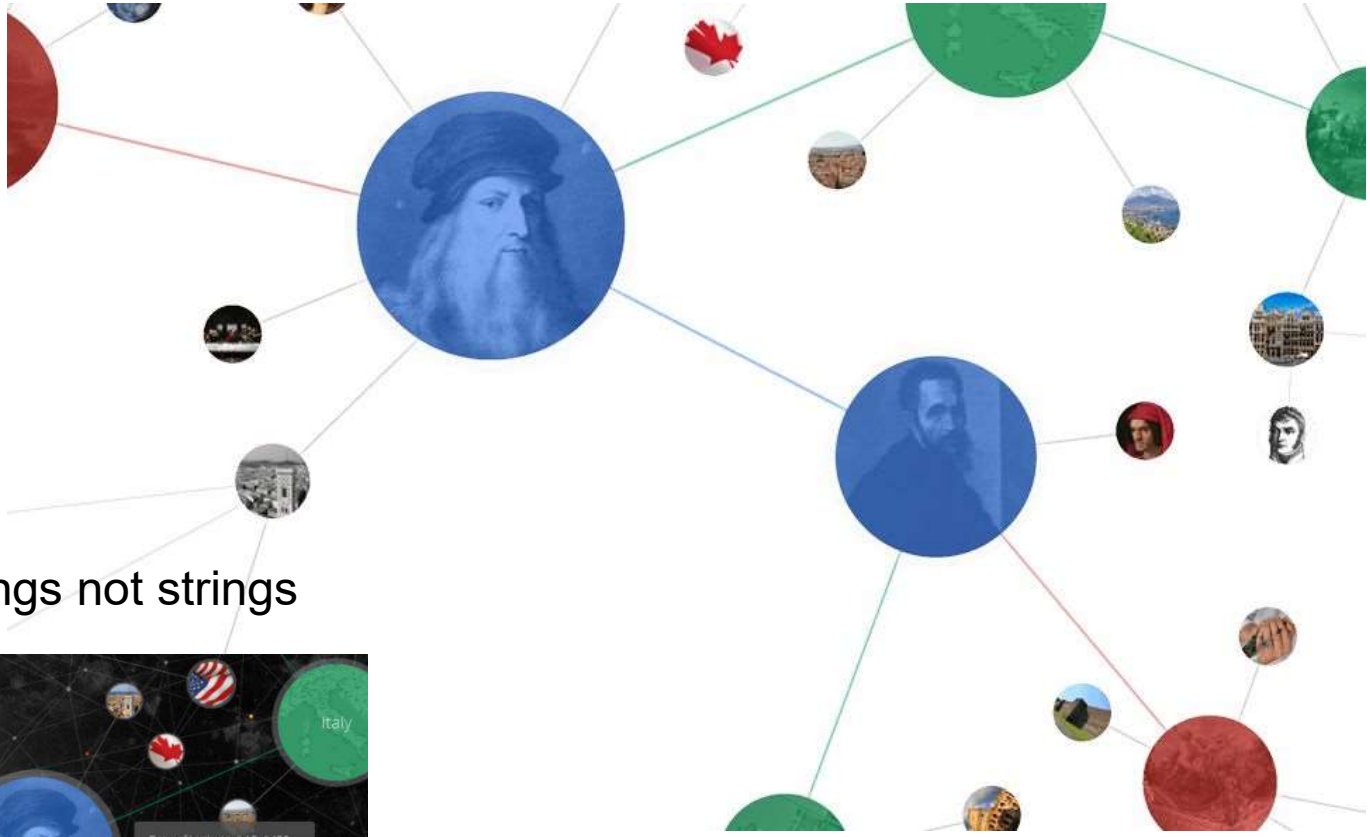
5 最短路径

6 拓扑排序

7 关键路径

jyang
@cs.ecnu.edu.cn

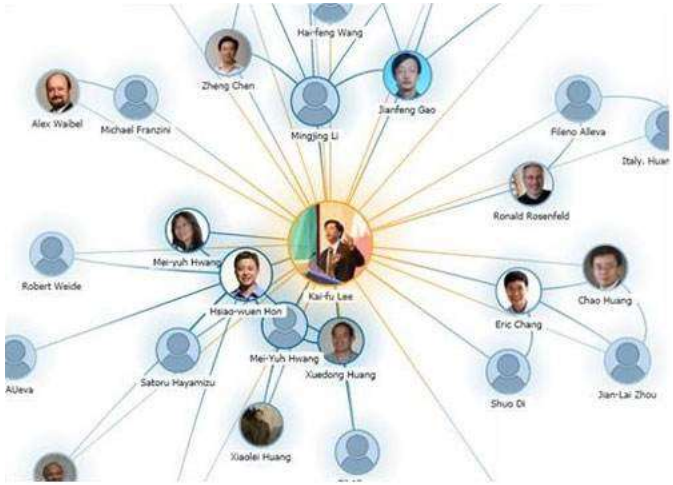
第七章　图

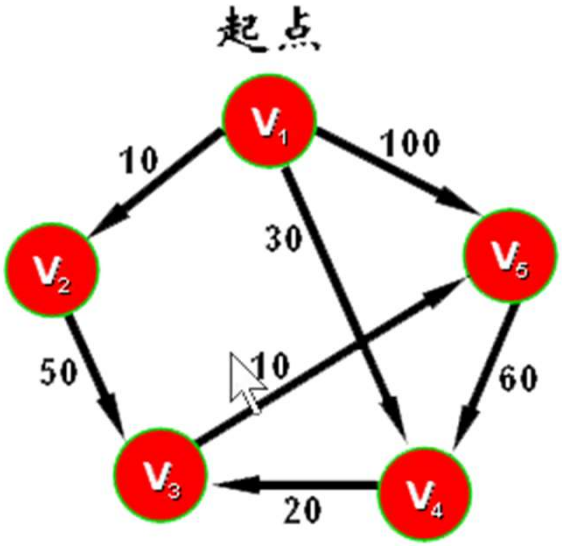


谷歌知识图谱: things not strings

jyang
@cs.ecnu.edu.cn

第七章　图





jyang
@cs.ecnu.edu.cn

第七章　图

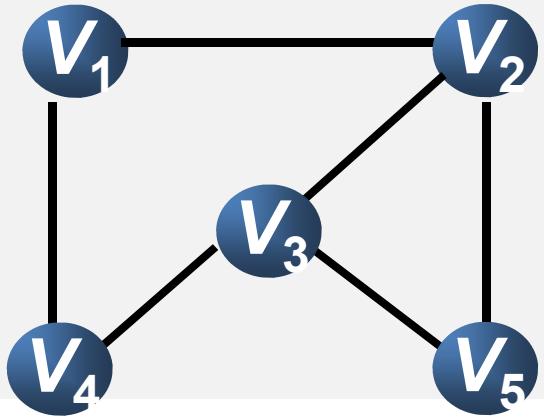
1 图的基本概念

无向图

一个无向图G是一个有序对（V，E），V是一个有限个顶点的集合，E是由V中两个不同元素组成的子集的集合，E中的元素称为边。

边：（v1, v2）（v1, v4）（v2, v3）（v2, v5）（v3, v4）（v3, v5）

相邻接：在无向图G中，如果i≠j，i，j∈V，（i, j）∈E，即i和j是G中两个不同的顶点，（i，j）是图G中的一条边，那么我们称顶点i和顶点j 相邻接。



顶点：vertex，(复数：vertices,vertexes）

jyang
@cs.ecnu.edu.cn

第七章　图

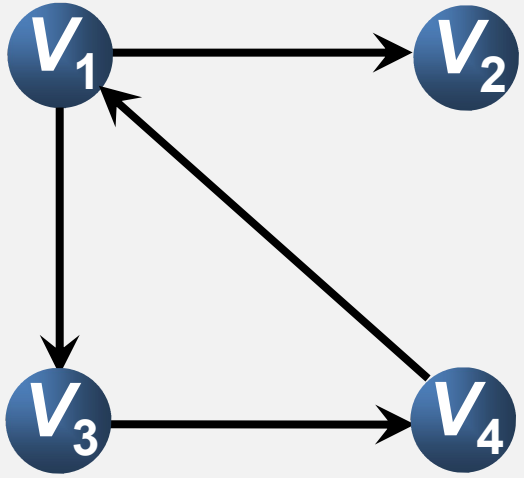
1 图的基本概念

有向图

一个有向图G是一个有序对（V，E），V是一个有限个顶点的集合，E是由V中两个不同元素组成的有序对的集合，E中的元素称为边。

边：〈v1, v2〉〈v1, v3〉〈v3, v4〉〈v4, v1〉

在有向图G中，如果i≠j，i，j∈V，〈i，j〉∈E，（即i和j是G中两个不同的顶点），〈i，j〉是图G中的一条边，那么我们称顶点i是这条边的尾顶点，j是这条边的头顶点。



第七章　图

1 图的基本概念

图中不能有一顶点到自身的边。

在无向图中，一对顶点之间不能有两条边。
在有向图中，一对顶点不能有相同方向的两条边，但可以有不同方向的两条边。

对于图G=(V,E)，我们可用V（G）表示图G的顶点集合，用E（G）表示图G的边的集合。
假设有两个图G和G’，同为无向图或同为有向图，
　　G=(V,E)，　G’=(V’,E’)，
且满足 $V' \subseteq V$，$E' \subseteq E$，则称G’是G的子图。

jyang
@cs.ecnu.edu.cn

完全无向图：每对顶点之间都有一条边的无向图，称为完全无向图。

完全有向图：每对顶点i和j之间都有边<i，j>和<j，i>的有向图，成为**完全有向图。**

具有n个顶点的完全无向图有多少条边？
n(n-1)/2条

具有n个顶点的完全有向图有多少条边？
n(n-1)条

jyang
@cs.ecnu.edu.cn

第七章　图

1 图的基本概念

路径：在无向图$G=(V, E)$中，从顶点v到顶点w之间的路径是一个由不同顶点组成的顶点序列$(v_0, v_1, \cdots\cdots, v_k)$，其中，$v_0=v$，$v_k=w$，$(v_i,v_{i+1})\in E$（$G$）（$0\le j<k$）。

用$(v_0, v_1, \cdots\cdots, v_k)$表示这条路径，这条路径的长度为k。

只有一个顶点时，称v到自身的路径长度为0。

若G是有向图，则路径也是有方向的，$<v_0, v_1, \cdots\cdots, v_k>$，其中，$v_0=v$，$v_k=w$，$<v_i,v_{i+1}>\in E$（$G$）（$0\le j<k$）

jyang
@cs.ecnu.edu.cn

第七章　图

1 图的基本概念

如果无向图G中每对顶点v和w都有从v到w的路径，那么称**无向图G是连通的**。无向图G中的极大连通子图为图G的**连通分量**。

如果有向图G中每对顶点v和w都有从v到w的路径，则称**有向图G是强连通的**。

如果有向图G中每对顶点v和w，有一个由不同顶点组成的顶点序列<v0, v1, ……, vk>，其中v0=v, vk=w, 且<vi, vi+1> ∈***E*** *或* <vi+1, vi> ∈***E*** （0≤i<k），那么称**有向图G是弱连通的**。

强连通的有向图一定是弱连通的。
有向图G的极大强连通子图为图G的**强连通分量**，
有向图G的极大弱连通子图为图G的**弱连通分量**。

jyang
@cs.ecnu.edu.cn

如果图G中有一条路径（ v0, v1, ……, vk ），且v0=vk，那么称这条路径为回路（或环）。

如果图G是无向图，那么称此回路为无向回路。如果图G是有向图，那么称此回路为有向回路。

一个图如果没有回路，则称该图是一个无回路的图。
一个连通的无回路的无向图可以定义一棵树。（任选一个顶点作为树的根，则可以确定一棵树）。

jyang
@cs.ecnu.edu.cn

定点的度（无向图中）：在无向图G中，如果v∈V（*G*），那么与v邻接的顶点个数称为顶点v的度。

定点的出度和入度（有向图中）：在有向图G中，如果v∈V（*G*），那么邻接到v的顶点个数为顶点v的入度，邻接于v的顶点个数为顶点v的出度。

jyang
@cs.ecnu.edu.cn

第七章　图

	无向图	有向图
G=（V，E）	顶点，边 图中不能有顶点到自身的边	
边	(Vi,Vj)	<Vi,Vj>
有边相连的两个顶点	相邻	头顶点，尾顶点
每对顶点之间都有边	完全无向图 N个顶点，边：n（n-1）/2	完全有向图 N个顶点，边：n（n-1）
	路径，路径长度	
如果每对顶点之间都有路径	连通的	强连通
	极大连通子图为图的连通分量	强连通的，强连通分量； 弱连通的，弱连通分量
回路（环）	无向回路	有向回路
顶点的边的情况	顶点的度	顶点的入度，出度

jyang
@cs.ecnu.edu.cn

2 图的存储结构

常用的两种图的存储结构：邻接矩阵和邻接表

邻接矩阵——无向图

如果G=（V，E）是具有n（n≥1）个顶点的无向图，那么它的邻接矩阵A是一个n×n阶矩阵，定义A为

$$A(i,j)=\begin{cases}1 & \text{若}(i,j)\in E(G)\\ 0 & \text{否则}\end{cases}$$

顶点i的度：$\sum_{j=1}^{n} A(i,j)$

无向图的邻接矩阵是对称的，可以只存入下三角或上三角。

存储空间需要 n（n+1）/2

jyang
@cs.ecnu.edu.cn

第七章　图

2　图的存储结构

邻接矩阵——有向图

如果G=（V，E）是具有n（n≥1）个顶点的有向图，那么它的邻接矩阵A是一个n×n阶矩阵，定义A为

$$A(i,j)=\begin{cases}1 & \text{若} <i,j> \in E(G)\\ 0 & \text{否则}\end{cases}$$

所需存储空间为n^2

顶点i的出度：

$\sum_{j=1}^{n} A(i,j)$　（邻接矩阵A的第i行之和）

顶点j的入度：

$\sum_{i=1}^{n} A(i,j)$　（邻接矩阵A的第j列之和）

用邻接矩阵表示具有n个顶点的图时（有向或无向），要确定图中有多少条边，所需要的时间？

O（n^2）

jyang
@cs.ecnu.edu.cn

2 图的存储结构

邻接表——有向图或无向图

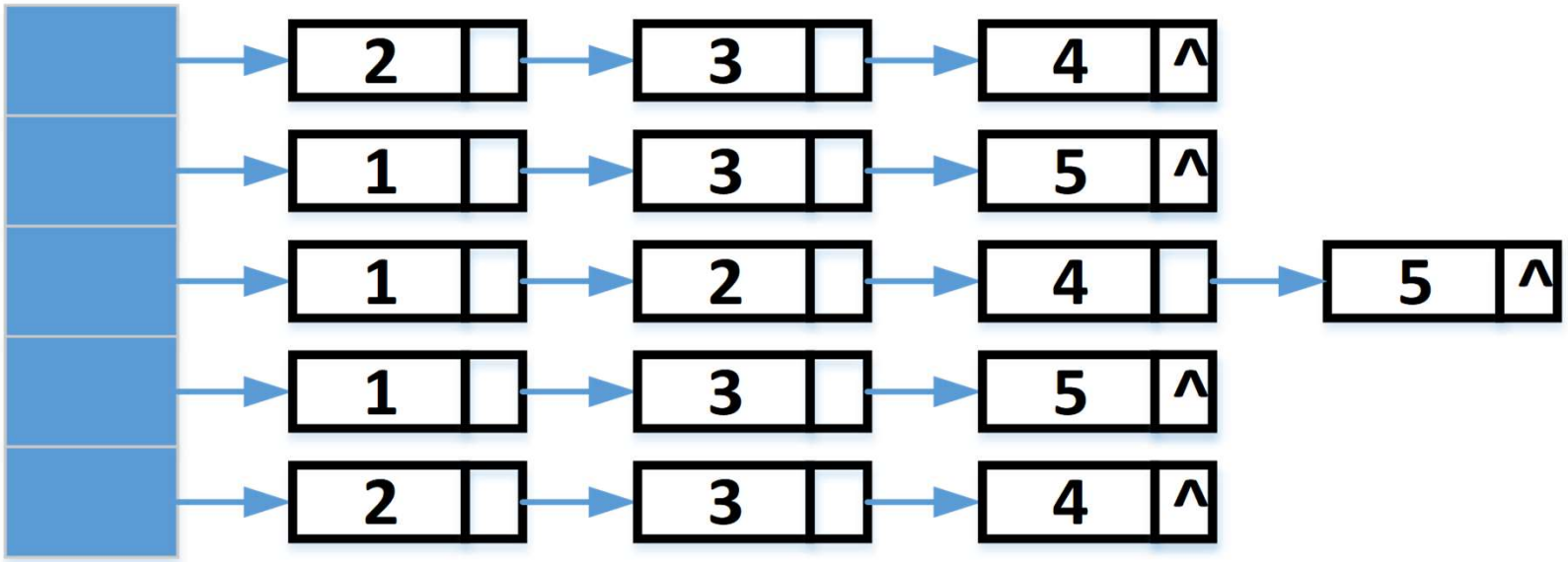
由n个链表组成，第i（1≤i≤n）个链表中的结点是由与顶点i相邻接（或是由邻接于顶点i）的顶点所构成。
N个链表的头指针通常按顺序方式进行存储，构成一个顺序表。

N个链表中可能会有某一个或者某几个是空的。

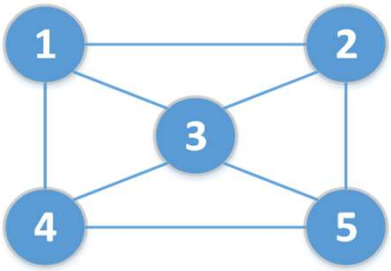
如果图G是无向图，有n个顶点和e条边，那么图G的邻接表需要由2e个结点组成的n个链表及由这n个链表的头指针组成的顺序表。
如果图G是有向图，有n个顶点和e条边，那么图G的邻接表需要由e个结点组成的n个链表及由这n个链表的头指针组成的顺序表。

jyang
@cs.ecnu.edu.cn

第七章　图

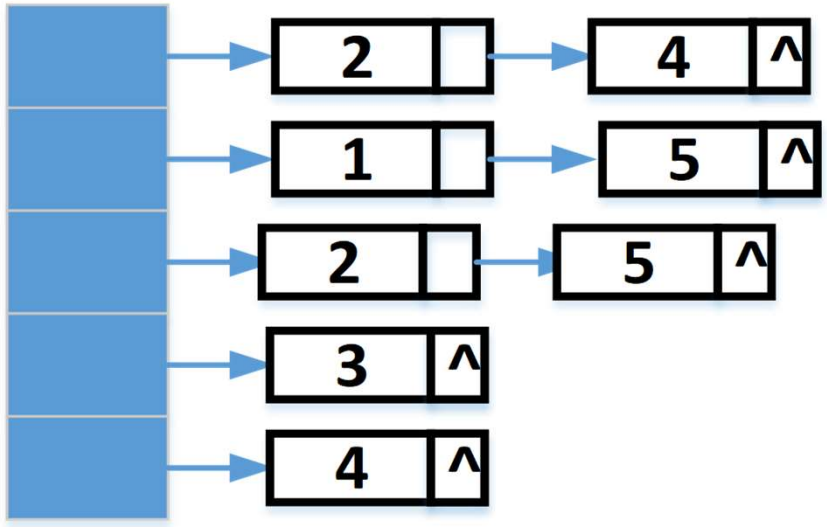


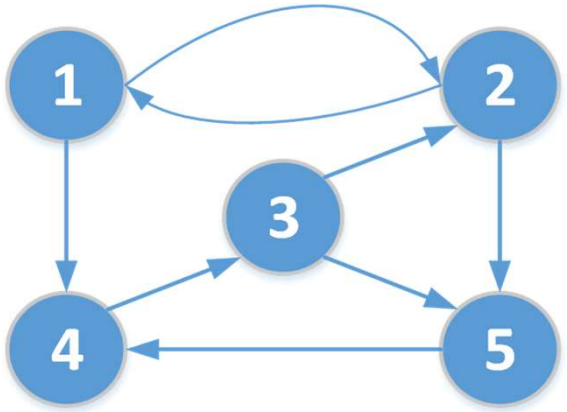

```
typedef struct link_node
{
    int   ver;
    struct link_node *link;

} L_NODE; //  邻接链表中的一个结点。
L_NODE *head[MAXN]
```



jyang
@cs.ecnu.edu.cn

第七章　图





邻接表

jyang
@cs.ecnu.edu.cn

2 图的存储结构

邻接表

在无向图的邻接表中，第i个链表的结点个数就是顶点i的度。

有向图：分为 出度、入度

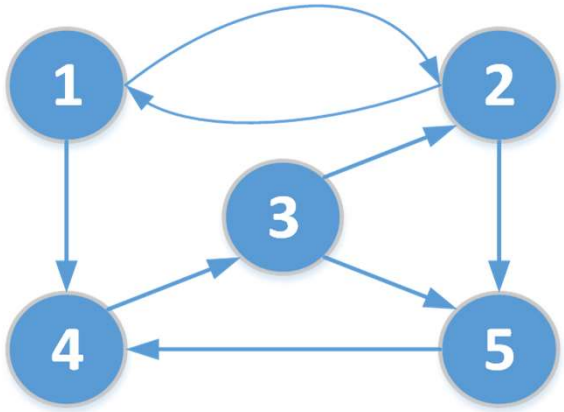
有向图，如何求有向图中结点i的入度？

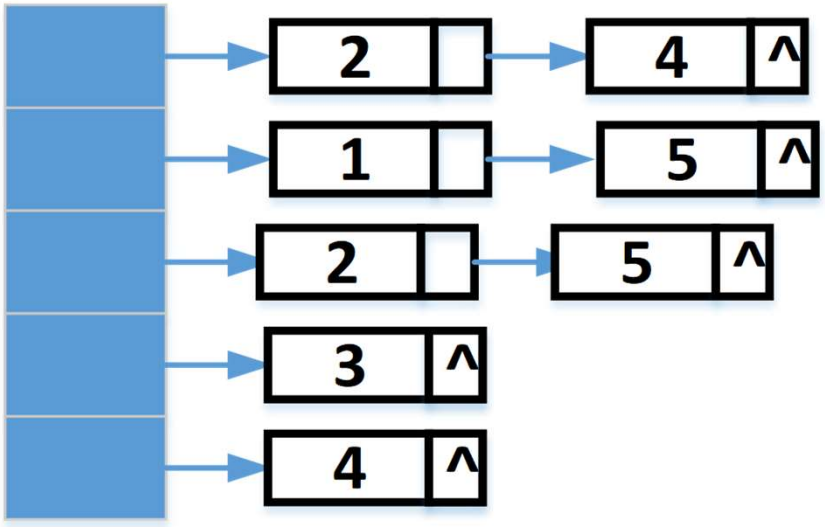
建立逆邻接表：把邻接到顶点i（1<=i<=n）的所有顶点构成逆邻接表中第i个链表。

如果用邻接表表示具有n个顶点e条边的图，那么确定图的边的数量所需的时间为O(n+e)

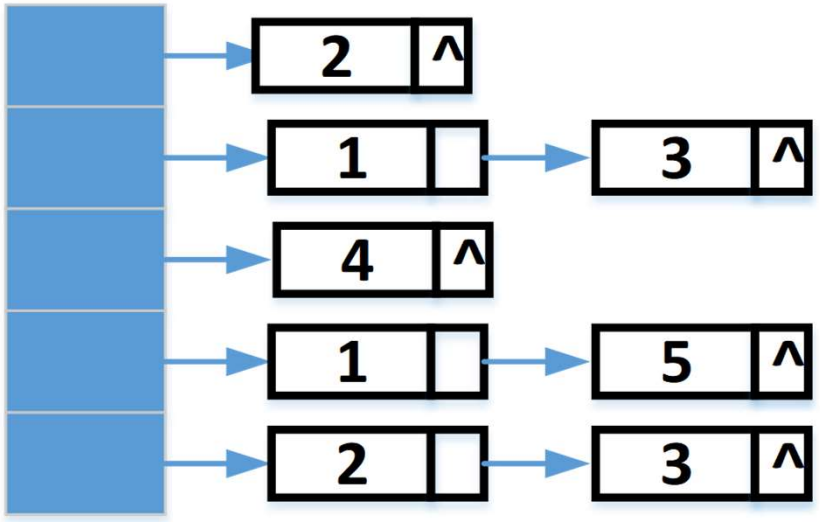
jyang
@cs.ecnu.edu.cn

第七章　图





邻接表



逆邻接表

jyang
@cs.ecnu.edu.cn

2 图的存储结构

邻接矩阵记录带权图：边上带权的图称为带权图。

边的权：表示从一个顶点到另一个顶点的距离或所花费的代价等。用w_{ij}表示边的权。

带权无向图的邻接矩阵A的两种表示方式：

$$A(i,j)=\begin{cases} w_{ij} & i\neq j \text{ 且 } (i,j)\in E(G) \\ 0 & \text{否则} \end{cases}$$

$$A(i,j)=\begin{cases} w_{ij} & i\neq j \text{ 且 } (i,j)\in E(G) \\ 0 & i=j \\ \infty & \text{否则} \end{cases}$$

带权有向图的邻接矩阵A的两种表示方式：

$$A(i,j)=\begin{cases} w_{ij} & i\neq j \text{ 且 } \langle i,j\rangle \in E(G) \\ 0 & \text{否则} \end{cases}$$

$$A(i,j)=\begin{cases} w_{ij} & i\neq j \text{ 且 } \langle i,j\rangle \in E(G) \\ 0 & i=j \\ \infty & \text{否则} \end{cases}$$

2 图的存储结构

图的存储结构：

邻接矩阵	存储空间和结点数量n有关（n^2）	方便求入度和出度
邻接表	存储空间和结点数量n、边的数量e有关	求入度：建立逆邻接表

3 图的遍历与求图的连通分量

当G是一个连通无向图时，从图G中任何一个顶点v出发，沿着G中的边访问该图中所有顶点，使每个顶点被访问且只被访问一次，我们称这一过程为图的遍历。

在图的遍历过程中，为了避免同一顶点被访问多次，可以用数组记录访问过的顶点。

遍历图的方法：

深度优先搜索法（DFS:Depth First Search ）
广度优先搜索法（BFS:Breadth First Search）

深度优先搜索法DFS

首先访问出发顶点v；
选择一个与v相邻接且未被访问过的顶点w访问之，再从w开始进行深度优先搜索；
每当到达一个其所有相邻接的顶点都已被访问过的顶点，就从最后所访问的顶点开始，依次退回到尚有邻接顶点未曾访问过的顶点u，并从u开始进行深度优先搜索。

这个过程进行到所有顶点都访问过，或从任何一个已访问的顶点出发，再也无法到达未曾访问过的顶点，则搜索过程结束。

深度优先搜索法——递归实现

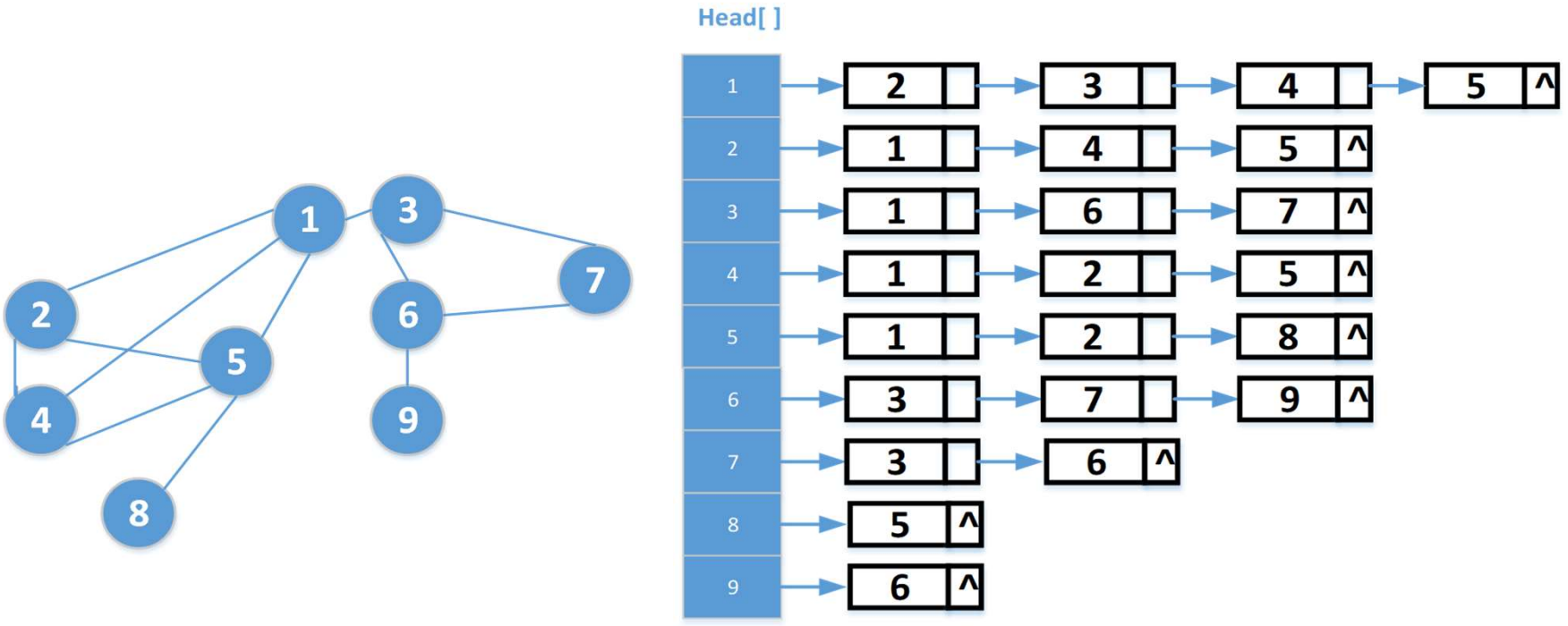
```
typedef struct link_node
{
    int   ver;
    struct link_node *link;

} L_NODE; //  邻接链表中的一个结点。
L_NODE *head[MAXN];

int visit [MAXN];

void init(int n)
{
        for(int i=1;i<=n;i++)  visit[i]=0;
}
```

深度优先搜索法——递归实现

```
void dfs(int u)  //递归实现深度优先搜索,邻接表存储
{
        L_NODE *t;
        visit[u]=1;  //表示u已经被访问过
        printf("%4d",u);//访问u
        t=head[u];  //找到和u邻接的结点的链表的头指针
        while(t!=NULL)
        {
            if(visit[t->ver]==0)   dfs(t->ver);
            //相邻结点未被访问过,
            则以这个结点为出发点进行深度优先遍历
            t=t->link; //访问下一个与u相邻接的结点
        }
}
```

jyang
@cs.ecnu.edu.cn

第七章　图



深度优先：dfs(1);
顶点序列：1,2,4,5,8,3,6,7,9
边: (1,2) (2,4) (4,5) (5,8) (1,3) (3,6) (6,7) (6,9)

深度优先搜索法——递归实现

n个顶点和m条边，因为DFS对邻接表中的每个结点至多检查一次，共有2m个结点，所以执行时间为O（n+m）。
如果图使用邻接矩阵表示，那么决定与某个顶点v邻接的所有顶点所需要的时间为O（n），因为有n个顶点要访问，所以执行时间为O（n^2）

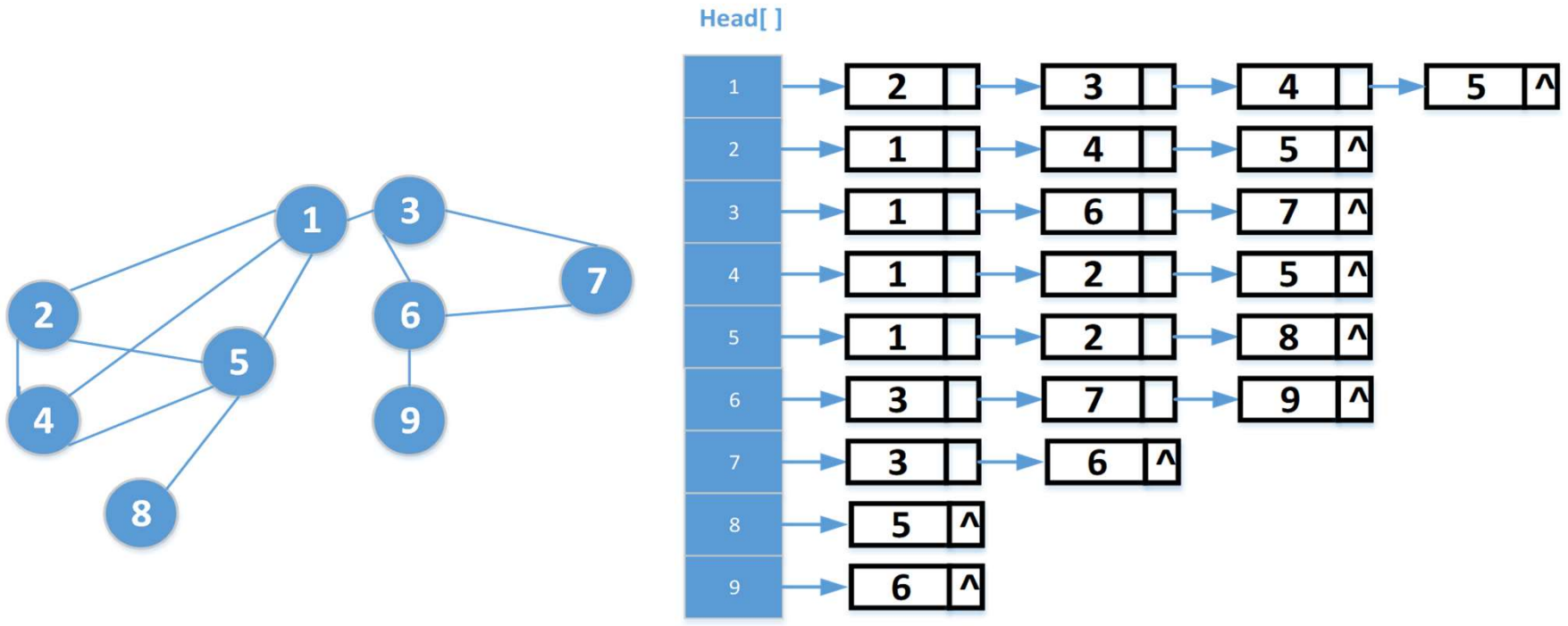
广度优先搜索法——BFS

首先访问出发顶点v；
然后访问与与顶点v邻接的全部顶点$w_1, w_2, \cdots, w_t$
再依次访问与$w_1, w_2, \cdots, w_t$ 邻接的全部顶点（已访问过的除外）
依次类推

直到图中所有顶点都被访问到，或者出发顶点v所在的连通分量的所有顶点都被访问到为止。

jyang
@cs.ecnu.edu.cn

第七章　图



首先访问出发顶点v；
然后访问与顶点v邻接的全部顶点$w_1, w_2, \cdots, w_t$
再依次访问与$w_1, w_2, \cdots, w_t$ 邻接的全部顶点（已访问过的除外）
依次类推，直到图中所有顶点都被访问到，或者出发顶点v所在的连通分量的所有顶点都被访问到为止。

广度优先：BFS(1);
顶点序列: 1,2,3,4,5,6,7,8,9
边: (1,2) (1,3) (1,4) (1,5) (3,6) (3,7) (5,8) (6,9)

jyang
@cs.ecnu.edu.cn

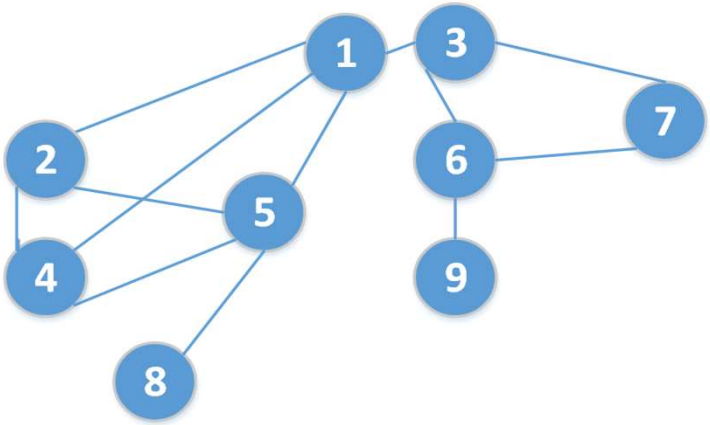
3 图的遍历与求图的连通分量

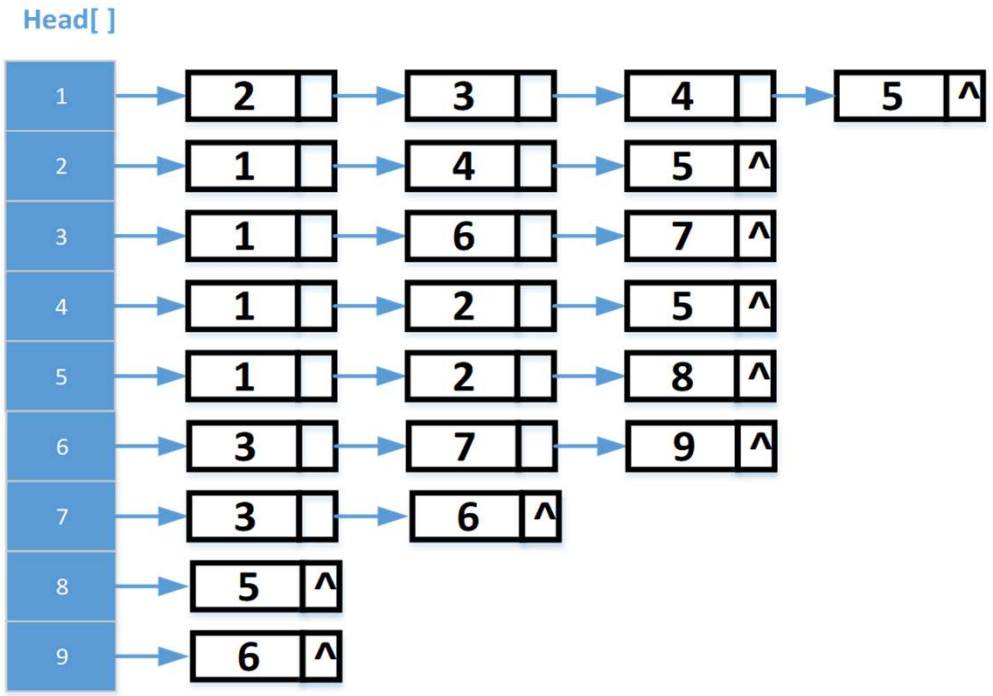
广度优先搜索法——利用队列实现广度优先搜索

```
void bfs(int u)
{……
        printf (…….., u) ;
        visit[u]=1;  //从结点u开始进行访问
        queue.qa=0;  //队首，qa标记出 可以出队的结点的下标，出队后+1
        queue.qe=0;  //队尾，标记出队列中的最后一个结点，入队前+1，
        queue.item[0]=u;//入队
        while(queue.qa<=queue.qe)//队列不空则继续
         {      v=queue.item[queue.qa++];  //取队首结点，出队
                t=head[v];//邻接表，取存以v相邻的顶点的链表的头放入t
                while(t!=NULL)//处理这条链上的所有结点，也就是”广度优先”访问
                {       w=t->ver;
                        if(visit[w]==0) //w没被访问过
                        {       printf("%4d",w);//访问w
                                visit[w]=1;  //标记为已访问
                                queue.item[++queue.qe]=w;  //入队，放于队尾
                        }
                        t=t->link; //访问下一个结点
                }
        }
}
```

jyang
@cs.ecnu.edu.cn

第七章　图





Qa：指向当前队首，出队后+1
Qe：指向当前队尾，入队前+1
队不空：qa<=qe

广度优先搜索法

以广度和深度优先搜索法遍历图所需要的时间的数量级是相同的O（n+m），（两种方法的差别在于搜索顶点的顺序不同）

用邻接矩阵表示：O（n^2）

求图的连通分量

无向图是连通的：从图中任意顶点出发遍历，可以访问所有顶点

无向图是非连通的：从图中任一顶点v出发遍历图，不能访问到该图的所有顶点，而只能访问到包含顶点v的极大连通子图（即连通分量）中的所有结点。

求图中所有连通分量：
对图的每个顶点进行检验。若被访问过，则该顶点落在已被求出的连通分量上；若未被访问过，则从该顶点出发遍历图，可求得图的另一个连通分量。在得到一个连通分量时，输出该连通分量所包含的顶点和边。

jyang
@cs.ecnu.edu.cn

4 生成树和最小代价生成树

生成树：设G是一个连通无向图，若G’是包含G中所有顶点的一个无回路的连通子图，则称G’为G的一棵生成树。

具有n个顶点的连通无向图至少有（n-1）条边，而其生成树G’恰好有（n-1）条边。在生成树G’中任意添加一条边，则会形成一个回路。

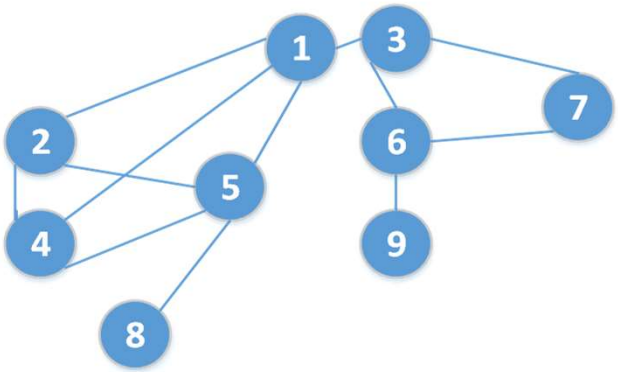
假设一个连通无向图G=(V,E), 边的集合为E(G),从任一顶点出发遍历，
E(G)可分成两个集合 T(G)和 B(G):
T(G)遍历过程经过的边的集合, B(G)剩余的边的集合。

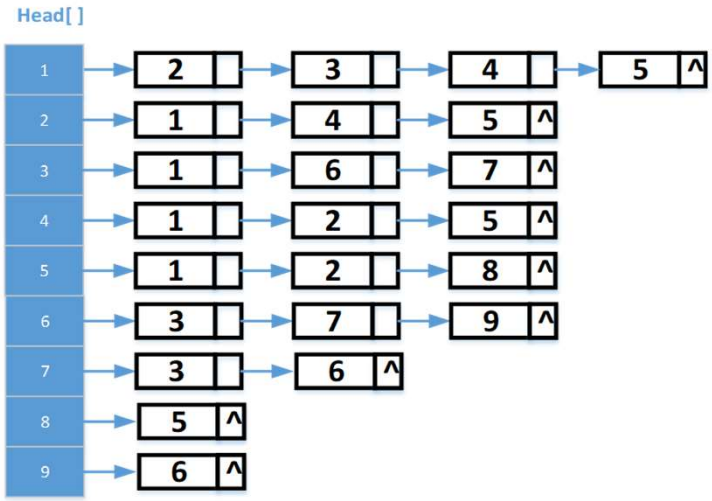
G’=(V,T(G)) 是G的一棵生成树。

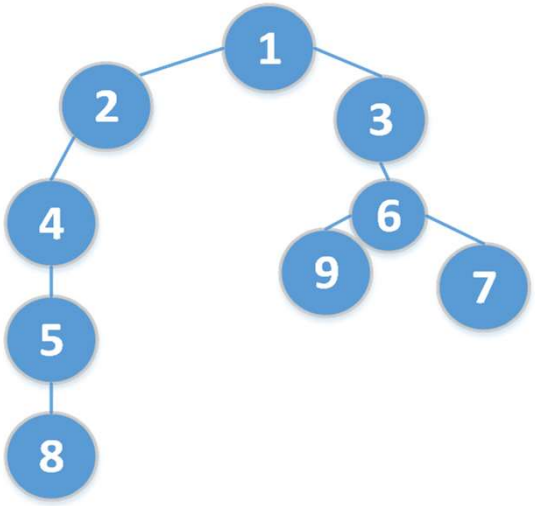
通过深度优先遍历得到的生成树叫深度优先生成树；通过广度优先遍历得到的生成树叫广度优先生成树。
连通无向图的生成树不是唯一的。

jyang
@cs.ecnu.edu.cn

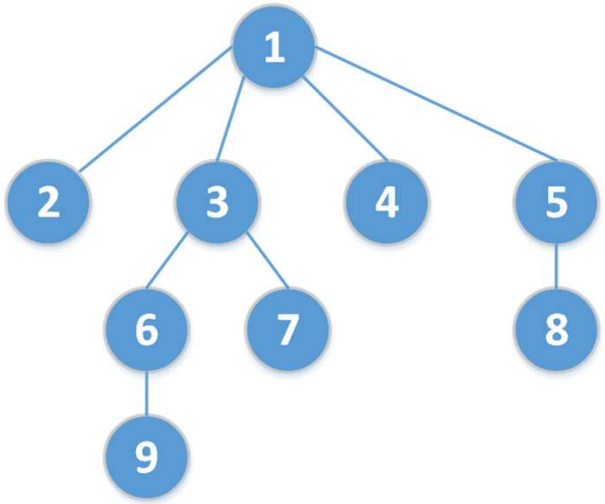
第七章　图







DFS（1）　得到的顶点序列和边
DFS生成树



BFS　（1）　得到的顶点序列和边
BFS生成树

4 生成树和最小代价生成树

最小代价生成树（Minimum Spanning Tree，MST）

例：用带权的连通无向图G的顶点表示城市，边表示连接两个城市之间的通信线路，边上的权表示线路的距离或代价。

若有n个城市，可修建构造n(n-1)/2条线路，每条线路都有权表示代价。如何在这些可能的线路中，选择其中（n-1）条线路使n个城市连通，使其总的代价最小（或者线路的总的长度最短）？

最小生成树MST：一个带权连通无向图G的最小（代价）生成树是G的所有生成树中边上权之和最小的一棵生成树。

Prim算法
Kruskal算法

jyang
@cs.ecnu.edu.cn

4 生成树和最小代价生成树

Prim算法

带权连通无向图 $G=(V,E)$，设U为V中构成MST的顶点集合，初始时 $U=\{u_0\}$

基本思想：

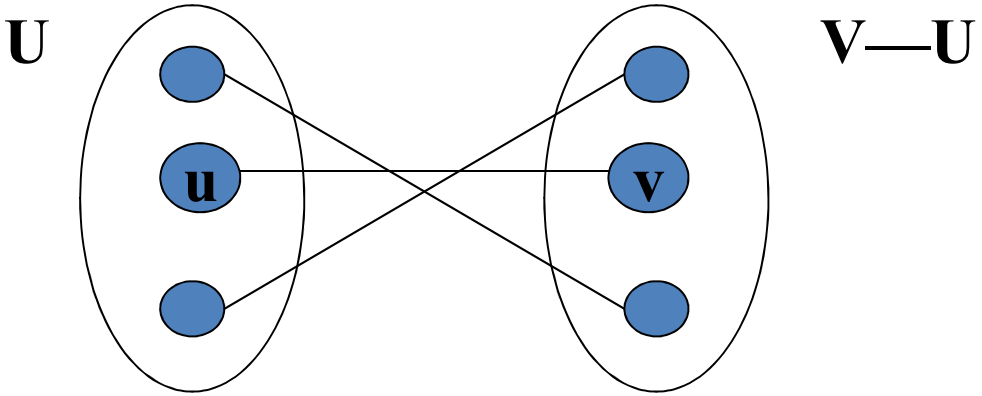
- 从带权连通无向图 $G=(V,E)$ 中的某一顶点 u_0 出发，选择与它关联的具有最小权值的边(u_0, v)，将其顶点加入到生成树的顶点集合U中。
- 以后每一步从一个顶点在U中，而另一个顶点不在U中的各条边中选择权值最小的边(u, v),把它的顶点加入到集合U中。如此继续下去，直到$U=V$为止。

每一步都会为一棵生长中的树添加一条边——一开始，这棵树只有一个顶点，共添加n-1条边，每次总是将下一条连接树中的顶点与不在树中的顶点且权重最小的边加入树中

Prim算法

证明 Prim算法的正确性：

设 G =（V，E）是一个带权连通无向图，T是一棵正在构造的最小生成树，U是T当前的顶点集合。若（u，v）是 G 中所有的 一个顶点在 U，另一个顶点不在U 的边中具有最小权值的一条边，则G中包含T的最小代价生成树一定包含边（u,v）。



Prim算法

证明 Prim算法的正确性（反证）：假设 G 中任何一棵最小生成树都不包含(u，v)。

设T是一棵最小生成树但不包含(u，v)。
由于T是最小生成树，所以 T 是连通的，因此有一条从u到v的路径，
且该路径上必有一条连接两个顶点集 U、V-U 的边(u',v')，
其中u'∈U，v'∈V-U。
当把边(u，v)加入到T中后，得到一个含有边(u，v)的回路。
删除边(u',v')，上述回路即被消除。
由此得到另一棵生成树 T'， T和T'的区别仅在于用边(u，v)代替了(u',v')。
由于(u，v)的权<= (u',v')的权，
所以， T'的权<= T的权，与假设矛盾。

Prim算法

Prim算法：从U只包含一个顶点，T为空开始，每一步加进去的都是最小生成树中应该包含的边。

在选择具有最小代价的边时，如果同时存在几条具有相同的最小代价的边，那么可以任选一条。

因此，构造的最小（代价）生成树可能不是唯一的，但代价总和是相等的。

Prim算法

（1）设T是带权连通无向图G=(V,E) 的最小（代价）生成树，初始时T为空，U为最小（代价）生成树的顶点集合，初始时U={v_0}，v_0是指定的某一个开始顶点。

（2）若U=V，则算法终止。否则，从E中选一条代价最小的边（u,v），使得u ∈U，v ∈V-U。将顶点v加到U中，将边（u，v）加到T中，转（2）。

例：图7.4.3

Prim算法
示例：
边：
(1,3) 1
(4,6)2
(2,5)3
(3,6)4
(2,3)5
(3,4)5
(1,4)5
(1,2)6
(3,5)6
(5,6)6

jyang
@cs.ecnu.edu.cn

4 生成树和最小代价生成树

Prim算法
示例：
从顶点1出发，
可以选择的边为

(1,3) 1
(4,6)2
(2,5)3
(3,6)4
(2,3)5
(3,4)5
(1,4)5
(1,2)6
(3,5)6
(5,6)6

Prim算法
示例：
选择边（1,3），
顶点集合 {1,3}
可以选择的边
(1,3) 1
(4,6)2
(2,5)3
(3,6)4
(2,3)5
(3,4)5
(1,4)5
(1,2)6
(3,5)6
(5,6)6

Prim算法
示例：
选择（3,6），
顶点集合 {1,3,6}
可以选择的边
(1,3) 1
(4,6)2
(2,5)3
(3,6)4
(2,3)5
(3,4)5
(1,4)5
(1,2)6
(3,5)6
(5,6)6

jyang
@cs.ecnu.edu.cn

4 生成树和最小代价生成树

Prim算法
示例：
选择（4,6），
顶点集合 {1,3,6,4}
可以选择的边
(1,3) 1
(4,6) 2
(2,5) 3
(3,6) 4
(2,3) 5
(3,4) 5
(1,4) 5
(1,2) 6
(3,5) 6
(5,6) 6

Prim算法
示例：
选择（2,3），
顶点集合 {1,3,6,4，2}
可以选择的边
(1,3) 1
(4,6) 2
(2,5) 3
(3,6) 4
(2,3) 5
(3,4) 5
(1,4) 5
(1,2) 6
(3,5) 6
(5,6) 6

Prim算法
示例：
选择（2,5），
顶点集合 {1,3,6,4，2,5}

jyang
@cs.ecnu.edu.cn

4 生成树和最小代价生成树

	[1]	[2]	[3]	[4]	[5]	[6]
[1]	0	6	1	5	∞	∞
[2]	6	0	5	∞	3	∞
[3]	1	5	0	5	6	4
[4]	5	∞	5	0	∞	2
[5]	∞	3	6	∞	0	6
[6]	∞	∞	4	2	6	0

Prim算法

实现：使用邻接矩阵的存储方式。

Lowcost数组：用于存放代价
（正在构建的MST的集合U到其他顶点的代价，
若顶点已经加入到U中，则代价为0）
closest数组，(closest[k],k)存放的是一条边

P212-Prim算法.pdf
N个顶点，函数主体 嵌套循环 时间复杂度为$O(n^2)$

jyang
@cs.ecnu.edu.cn

4 生成树和最小代价生成树

Kruskal算法

初始状态：n棵单顶点的树，不断将两棵树合并（用可以找到的“最短”边），直到只剩下一棵树，它就是最小生成树。

（1）初始时，T=（V，　Φ），即T有n个连通分量组成，每个连通分量只有一个顶点，没有边。
（2）把边的权值按照从小到大排序，依次进行如下选择：若当前被选择的边的两个顶点在不同的连通分量中，则把这条边加到T中（即每次选权最小且不构成回路的一条边加入T），直到T中有(n-1)条边为止。
例：P214

4 生成树和最小代价生成树

Kruskal算法

```
Sort E edges by increasing weight
T = { }

for (j = 0; j < edgeList.length; j++)
  if adding e = edgelist[j] does not form a cycle
    add e to T
  else ignore e

MST = T
//算法核心：判断当前被选择的边的两个顶点在不同的连通分量中
```

Kruskal算法

//算法核心：判断当前被选择的边的两个顶点是否在不同的连通分量中

实现方法：
将在同一个连通分量中的顶点看成一个集合。
判断候选边的两个顶点是否在不同的集合。
如果在同一个集合，放弃这条边。
如果在不同的集合，选择这条边，同时把两个集合合并。

选用什么结构来完成？
Union-find,方便实现集合的查找和合并。

4 生成树和最小代价生成树

Kruskal算法

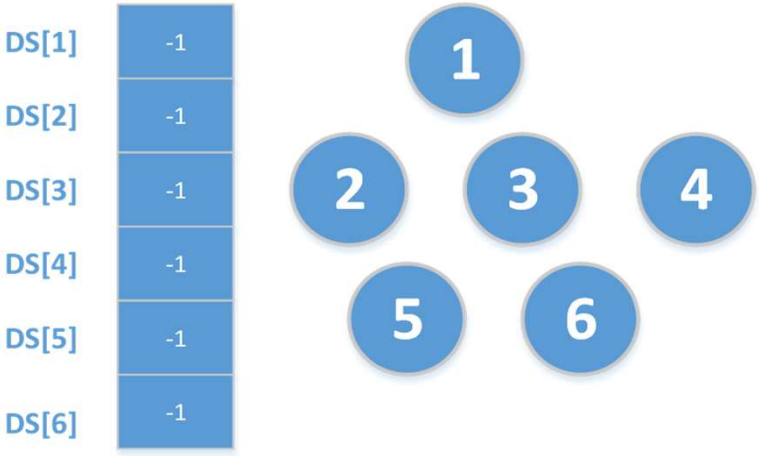
```
{.............
        for (i..verNum) ds[i]=-1;
        Sort E edges by increasing weight
        for (j = 0; edgesAccepted<VerNum-1 && j<edgeList.length; j++)
        {
                u=find_Set (ds, E[j].begin) ;
                v=find_Set (ds, E[j].end) ;
                if (u!=v) {
                        ++edgesAccepted;
                        union_Set(ds,u,v) ;//执行union操作
                        //将符合条件的边加入到MST，此处输出
                        print(E[j].begin, E[j].end, E[j].weight)
                        }
        }
}
```

第七章 图

4 生成树和最小代价生成树

Kruskal算法

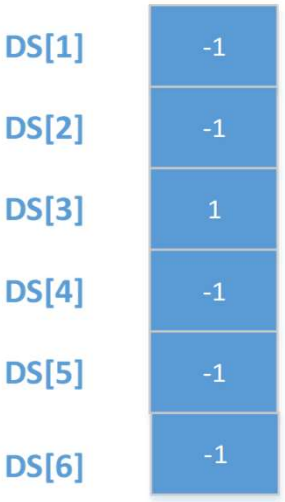
//算法核心：判断当前被选择的边的两个顶点是否在不同的连通分量中

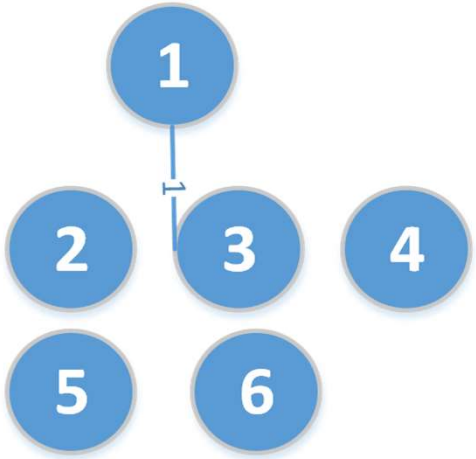


Sorted E edges (begin, end, weight)
(1,3,1)
(4,6,2)
(2,5,3)
(3,6,4)
(1,4,5)
(2,3,5)
(3,4,5)
(1,2,6)
(3,5,6)
(5,6,6)

```
for (j = 0; edgesAccepted<VerNum-1 && j<edgeList.length; j++)
{       u=find_Set (ds, E[j].begin) ;
        v=find_Set (ds, E[j].end) ;
        if (u!=v) {     ++edgesAccepted;
                        union_Set(ds,u,v) ;//执行union操作
                        print(E[j].begin, E[j].end, E[j].weight)
                  }
}
```

执行find_Set后， u=1,v=3,
向MST中 添加 （1,3,1）
union_Set(ds,u,v) 后 Ds[3]的值变为1



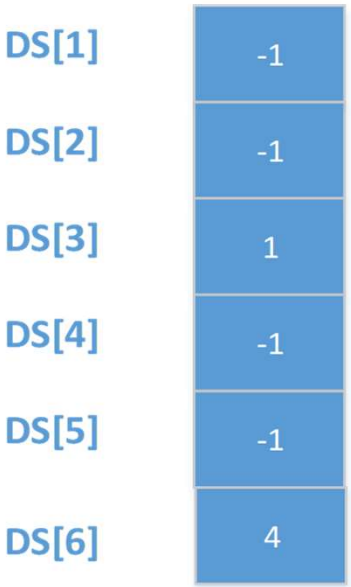


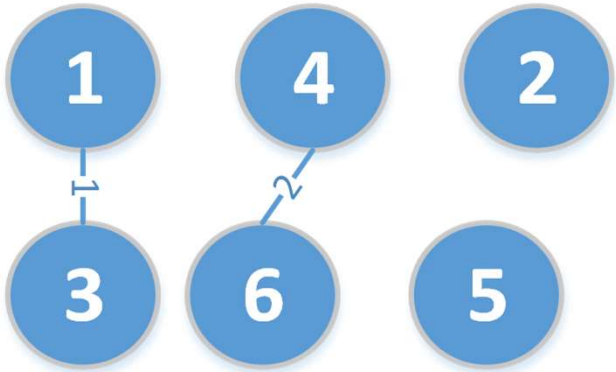
Sorted E edges (begin, end, weight)
(1,3,1)
(4,6,2)
(2,5,3)
(3,6,4)
(1,4,5)
(2,3,5)
(3,4,5)
(1,2,6)
(3,5,6)
(5,6,6)

jyang
@cs.ecnu.edu.cn

4 生成树和最小代价生成树

Kruskal算法



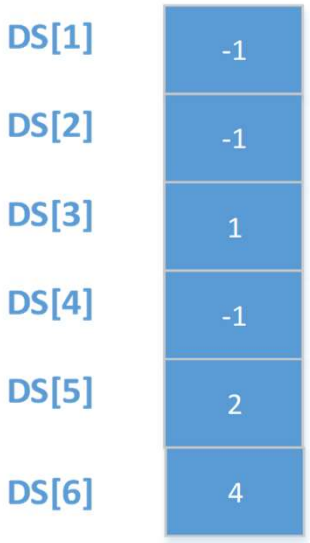


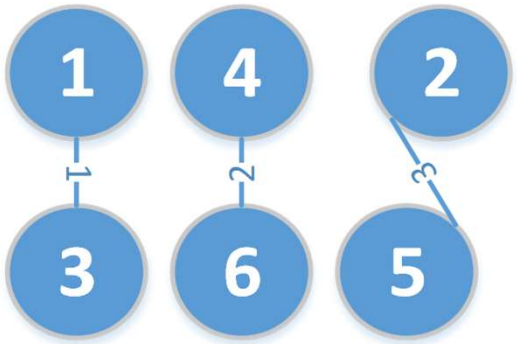
执行find_Set后，u=4,v=6,
添加 （4,6,2）
Ds[6]的值变为4

Sorted E edges (begin, end, weight)
(1,3,1)
(4,6,2)
(2,5,3)
(3,6,4)
(1,4,5)
(2,3,5)
(3,4,5)
(1,2,6)
(3,5,6)
(5,6,6)

4 生成树和最小代价生成树

Kruskal算法





执行find_Set后，u=2,v=5,
添加 （2,5,3）
Ds[5]的值变为2

Sorted E edges (begin, end, weight)
(1,3,1)
(4,6,2)
(2,5,3)
(3,6,4)
(1,4,5)
(2,3,5)
(3,4,5)
(1,2,6)
(3,5,6)
(5,6,6)

Kruskal算法

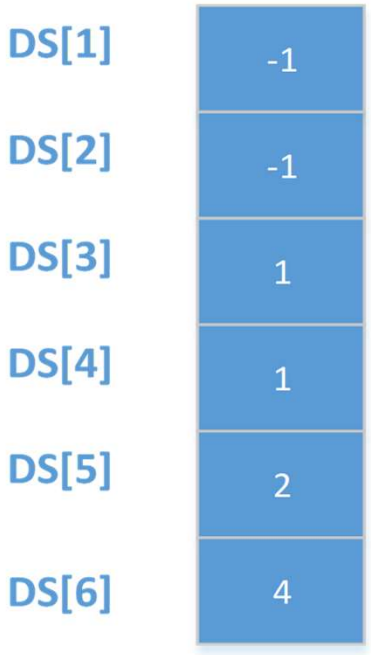
DS[1]	-1
DS[2]	-1
DS[3]	1
DS[4]	-1
DS[5]	2
DS[6]	4

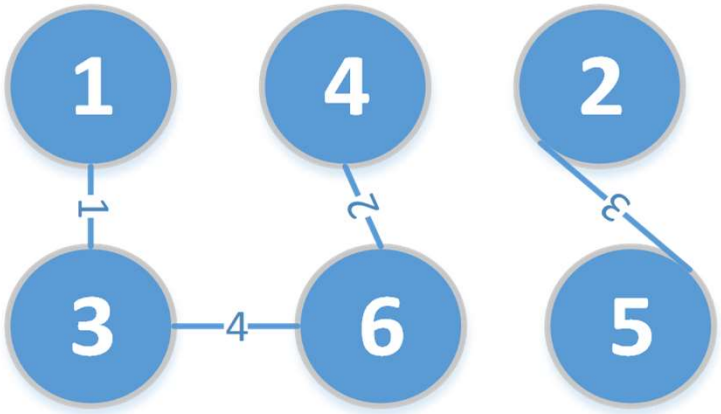
Sorted E edges (begin, end, weight)
(1,3,1)
(4,6,2)
(2,5,3)
(3,6,4)
(1,4,5)
(2,3,5)
(3,4,5)
(1,2,6)
(3,5,6)
(5,6,6)

执行u= find_Set （ds,3）后， u=1
执行v= find_Set （ds,6）后， v=4
添加 （3,6,4）
执行union_Set(ds,1,4) ;后 Ds[4]的值变为1

4 生成树和最小代价生成树

Kruskal算法

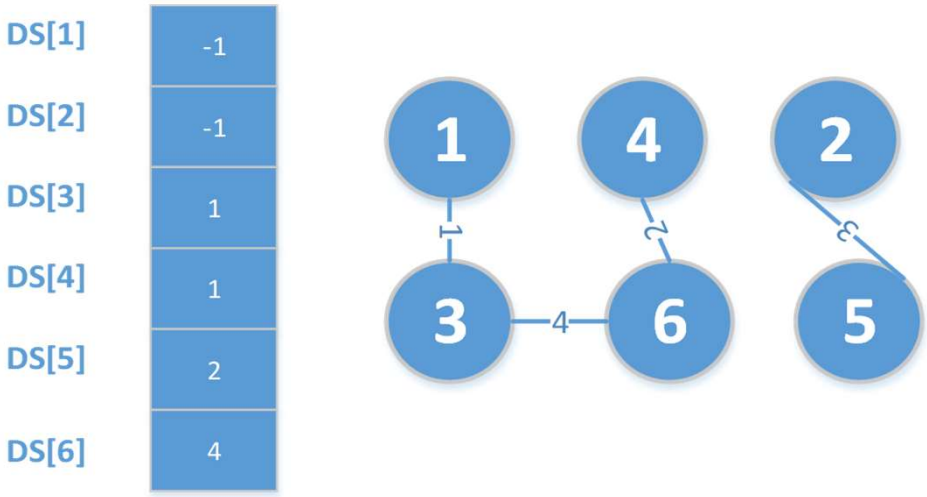




第七章　图

4 生成树和最小代价生成树

Kruskal算法



执行u= find_Set （ds,1）后，u=1
执行v= find_Set （ds,4）后，v=1

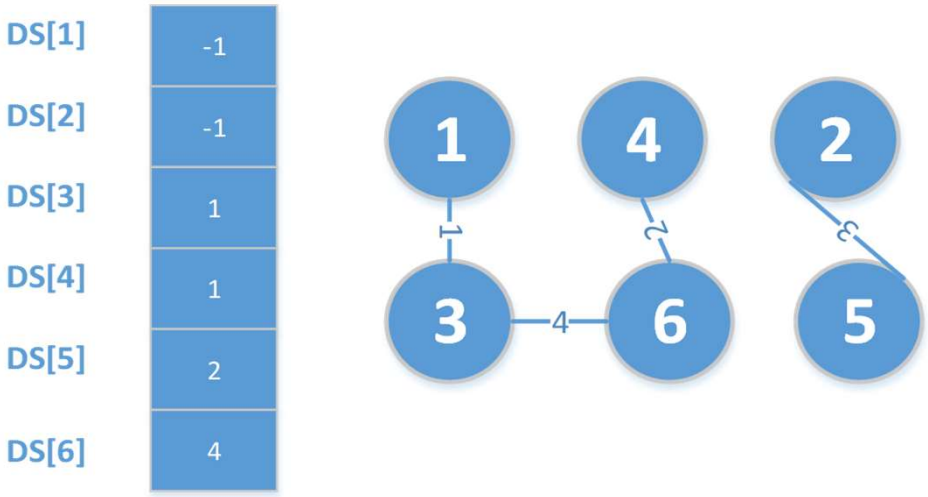
u=1,v=1,放弃当前候选边（1,4,5）

Sorted E edges (begin, end, weight)
（1,3,1）
（4,6,2）
（2,5,3）
（3,6,4）
（1,4,5）不被选中
（2,3,5）
（3,4,5）
（1,2,6）
（3,5,6）
（5,6,6）

第七章　图

4 生成树和最小代价生成树

Kruskal算法

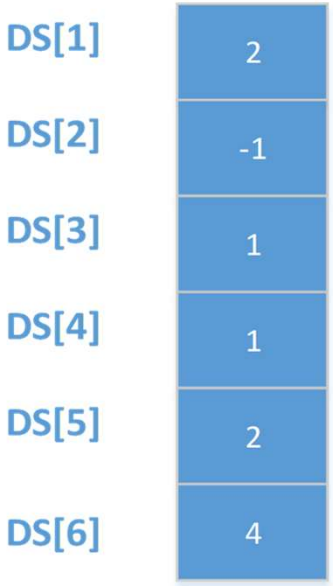


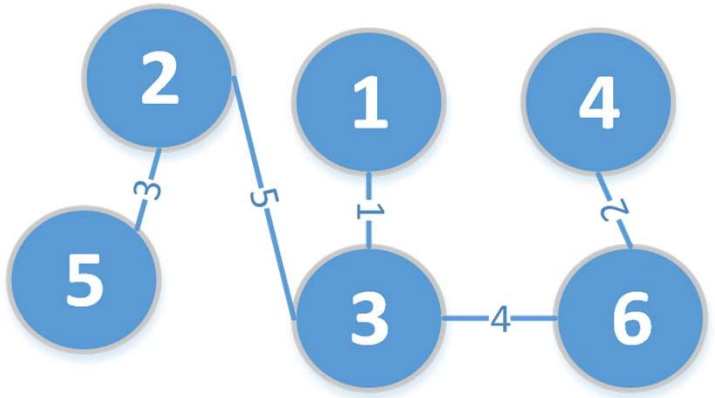
Sorted E edges (begin, end, weight)
(1,3,1)
(4,6,2)
(2,5,3)
(3,6,4)
(1,4,5) 不被选中
(2,3,5)
(3,4,5)
(1,2,6)
(3,5,6)
(5,6,6)

执行u= find_Set （ds,2）后，u=2
执行V= find_Set （ds,3）后，v=1
添加 （2,3,5）
执行union_Set(ds,2,1)后设ds[1]值为2

4 生成树和最小代价生成树

Kruskal算法





Kruskal算法

结论：带权连通无向图G有n个顶点，m条边

Kruskal算法求最小（代价）生成树的时间复杂度为O（$mlog_2m$）

Prim算法，时间复杂度为O(n^2)

适用性：

prim，　R.Prim在1961年给出的，“稠密”图的最小生成树算法。

Kruskal，“稀疏”图

5 最短路径

问题的提出：
用带权有向图表示交通网络，顶点表示城市，边<u, v>表示城市u到城市v的直通道路，权表示道路长度或花费时间等。人们通常会提出如下问题：
（1）两地之间是否有公路可通。
（2）在几条公路可通的情况下，哪一条代价最小？

约定：权是正的。路径的开始顶点为源点，路径的最后的顶点为终点。路径的长度是该路径上各边的权之和。

求某个顶点到其他顶点的最短路径——Dijkstra算法
求每一对顶点之间的最短路径——Floyd算法

jyang
@cs.ecnu.edu.cn

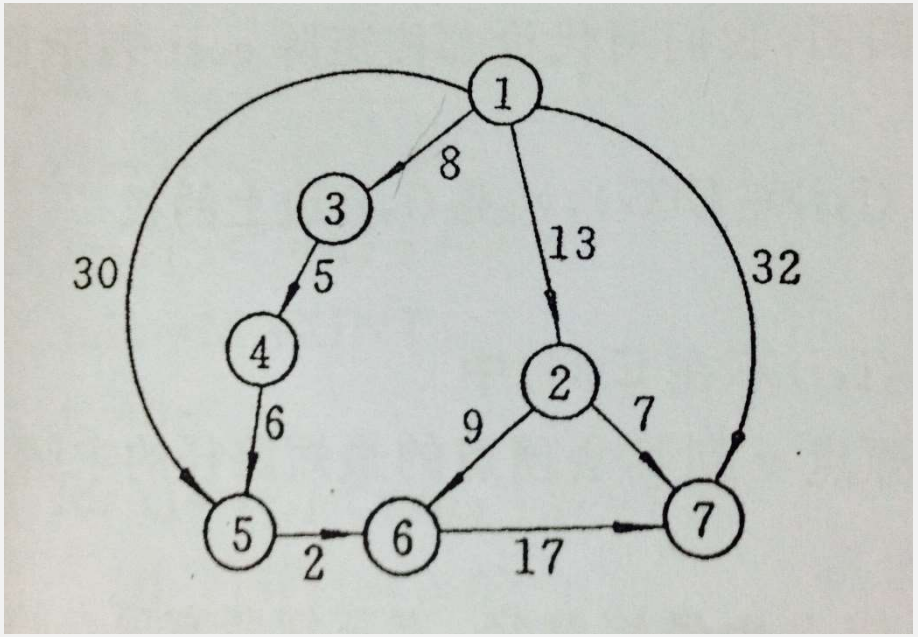
5 最短路径

求某个顶点到其他顶点的最短路径——Dijkstra算法

设图G=（V,E）是一个带权有向图，v是图G中指定的源点，在G中找出从v到其他顶点的最短路径。

例：以“1”为源点

<1,2>　13
<1,3>　8
<1,3，4> 13
<1,5>
<1,6>
<1,7>



5 最短路径

求某个顶点到其他顶点的最短路径——Dijkstra算法

Dijkstra算法：按路径长度不减次序产生最短路径的算法。

把图中的顶点集合V分成两组，令S表示已求出最短路径的顶点集合，其余尚未确定最短路径的顶点集合为第二组。按最短路径长度递增次序，逐个把第二组的顶点加入S中，直到从v出发可以到达的所有顶点都在S中。

5 最短路径

求某个顶点到其他顶点的最短路径——Dijkstra算法

初始时：S={v}，v到v的距离为零；第二组包含其他所有顶点，距离为：若图中有边<v,w>，则w的距离既是这条边上的权。否则，记为为∞；
每次从第二组的顶点中选取一个距离最小的顶点k，把k加入S中，并对第二组的各个顶点j的距离进行一次修改——若加入k做中间顶点，从v到j的距离比原来不经过顶点k的距离短，则修改j的距离值。
如此继续，直到G中所有的顶点都在S中，或再也没有可加入S的顶点存在。

5 最短路径

求某个顶点到其他顶点的最短路径——Dijkstra算法

P217

（1） S={v},

（2）按下列步骤逐个求得从v到其他顶点的最短路径，直至把所有顶点的最短路径都求出:

(a)选取不在S中且具有最小距离的顶点k

(b)把k加入集合S

（c）修改不在S中的顶点的距离

jyang
@cs.ecnu.edu.cn

第七章　图

	【1】	【2】	【3】	【4】	【5】	【6】	【7】
【1】	0	13	8	*	30	*	32
【2】	*	0	*	*	*	9	7
【3】	*	*	0	5	*	*	*
【4】	*	*	*	0	6	*	*
【5】	*	*	*	*	0	2	*
【6】	*	*	*	*	*	0	17
【7】	*	*	*	*	*	*	0

	S[ ]							dist[ ]							pre[ ]						
从顶点1出发	1	0	0	0	0	0	0	0	13	8	*	30	*	32	0	1	1	0	1	0	1
K=3 min=8	1	0	1	0	0	0	0	0	13	8	13	30	*	32	0	1	1	3	1	0	1
K=2 min=13	1	1	1	0	0	0	0	0	13	8	13	30	22	20	0	1	1	3	1	2	2
K=4 min=13	1	1	1	1	0	0	0	0	13	8	13	19	22	20	0	1	1	3	4	0	1
K=5 min=19	1	1	1	1	1	0	0	0	13	8	13	19	21	20	0	1	1	3	4	5	2
K=7 min=20	1	1	1	1	1	0	1	0	13	8	13	19	21	20	0	1	1	3	4	5	2
K=6 min=21	1	1	1	1	1	1	1	0	13	8	13	19	21	20	0	1	1	3	4	5	2

初始时：S={v}，v到v的距离为零；第二组包含其他所有顶点，dist[]：若图中有边<v,w>，则w的距离既是这条边上的权。否则，∞；
从第二组的顶点中选取一个距离最小的顶点k，
(1)把k加入S中，(2)并对第二组的各个顶点j的距离进行一次修改
如此继续，直到G中所有的顶点都在S中，或再也没有可加入S 的顶点存在。

5 最短路径

求某个顶点到其他顶点的最短路径——Dijkstra算法

结论：时间复杂度为 $O(n^2)$

问题：
1. 求解完成后，如何输出从源点v到其他顶点的最短路径？
2. 什么情况下，可以判断从源顶点出发，到另一个顶点的最短路径不止一条？

5 最短路径

求每一对顶点之间的最短路径

方法一:把带权有向图G中n个顶点的每一个顶点作为源点,重复执行Dijsktra算法n次。时间复杂度O(n^3)。

Floyd算法（时间复杂度为O(n^3)）：递推地产生一个矩阵序列。

问题：如何计算任意两个结点间最短路径的条数？

求每一对顶点之间的最短路径——Floyd算法

思想：假设求从顶点v_i到v_j的最短路径。如果从v_i到v_j有边，则从v_i到v_j存在一条长度为cost[i][j]的路径，但该路径不一定是最短路径。因为可能存在一条从v_i到v_j，但包含有其它顶点为中间点的路径。因此，需进行n次试探，测试从v_i到v_j能否有以顶点v_0，v_1，v_2，…，v_{n-1}为中间点的更短路径。

使用矩阵序列来完成递推过程。

$A^{(0)}$ 代价邻接矩阵

$A^{(k)}$（i，j）表示从顶点i到顶点j的中间顶点序号不大于k的最短路径的长度。

5 最短路径

求每一对顶点之间的最短路径——Floyd算法

$$\begin{cases} A^{(0)}(i,j) = cost(i,j) \\ A^{(k)}(i,j) = min\{A^{(k-1)}(i,j), A^{(k-1)}(i,k) + A^{(k-1)}(k,j)\} \\ 1 \le k \le n \end{cases}$$

P 220

jyang
@cs.ecnu.edu.cn

5 最短路径

求每一对顶点之间的最短路径——Floyd算法

```
void floyd(cost,int n)
{       int i,j,k;
        for(i=1;i<=n;i++)    //赋初值
          for(j=1;j<=n;j++)   {       a[i][j]=cost[i][j];
                                      path[i][j]=0;   }
        //嵌套循环，递推求A（k）
        for(k=1;k<=n;k++)
            for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                      if(a[i][k]+a[k][j]<a[i][j])
                              { a[i][j]=a[i][k]+a[k][j];
                                path[i][j]=k;
                              }
}
```

时间复杂度：O（n^3）

5 最短路径

求每一对顶点之间的最短路径——Floyd算法

P221

```
Cost
0 10 5
4 0  *
* 2  0
```

5 最短路径

传递闭包

求每一对顶点间的最短路径问题，首先必须确定有向图G中从顶点i到顶点j是否存在路径（路径长度为该路径所经过的边的数目）。

设A为有向图的邻接矩阵，A的传递闭包A^+ 可表示这种情况。由A求A^+ 的算法可以借鉴floyd算法来完成。

A*:自反传递闭包（认为自己到自己也存在一条路径）

jyang
@cs.ecnu.edu.cn

第七章　图

$$A=\begin{bmatrix}0&1&0&0\\1&0&1&0\\0&0&0&1\\0&0&0&0\end{bmatrix}$$

$$A^{+}=\begin{bmatrix}1&1&1&1\\1&1&1&1\\0&0&0&1\\0&0&0&0\end{bmatrix}$$

$$A^{*}=\begin{bmatrix}1&1&1&1\\1&1&1&1\\0&0&1&1\\0&0&0&1\end{bmatrix}$$

图7.5.6　G,A,A^{+}和 A^{*}

传递闭包

.....

```
for(k=1;k<=n;k++)   //递推求A（k）
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(a[i][j]==0)
                if ( a[i][k]+a[k][j]==2)
                    a[i][j]==1;
```

5 最短路径

将传递闭包A+修改为自反传递闭包A*

.....

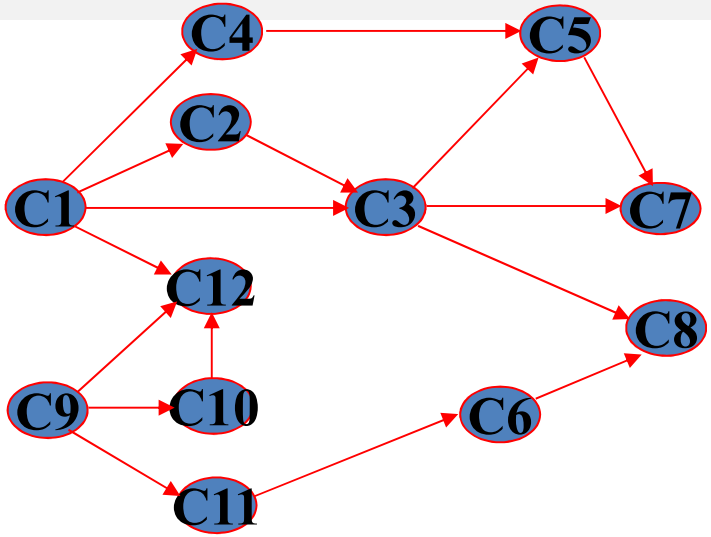
```
for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
      if(i==j && a[i][j]==0)   a[i][j]==1;
```

jyang
@cs.ecnu.edu.cn

6 拓扑排序

例：

课程代号	课程名称	先修课
C1	程序设计基础	无
C2	离散数学	C1
C3	数据结构	C1,C2
C4	汇编语言	C1
C5	语言的设计和分析	C3,C4
C6	计算机原理	C11
C7	编译原理	C3.C5
C8	操作系统	C3.C6



可以用有向图来表示，把这些课程看成一个个顶点，称之为活动(Activity)。
如果从顶点V_i到V_j之间存在有向边$< V_i, V_j >$,则表示活动i必须先于活动j进行。
这种图称做顶点表示活动的网络（Activity On Vertex network 简称AOV网)。

jyang
@cs.ecnu.edu.cn

6 拓扑排序

设S是一个集合，R是S上的一个关系，a,b是 S 中的元素，
若（a,b）∈ R ，
则称a是b关于R的**前趋元素**，b是a关于R的**后继元素**。

设S是一个集合，R是S上的一个关系，a,b,c是S中的元素，若有（a,b）∈ R和（b，c）∈ R，则必有（a,c）∈ R，那么我们称R是S上的一个**传递关系**。

若对于S中的任一元素 a，不存在（a,a）∈ R，则称R是S上的一个**非自反关系**。

若S上的一个关系 R 是传递的和非自反的，则称R是S上的一个**半序关系**。

6 拓扑排序

在任何一个具有半序关系R的有限集合中，至少有一个元素没有前趋，也至少有一个元素没有后继。

若R是集合S上的一个半序关系，A= $a_1, a_2, \cdots\cdots, a_n$是S中元素的一个序列，且当（$a_i, a_j$）$\in$ R 时，　有i<j，则称A是相对于R的一个拓扑序列。

如果a_i和a_j关于R毫无关系，那么它们在A中的排列次序不受限制。

我们称获得拓扑序列的过程为拓扑排序。

jyang
@cs.ecnu.edu.cn

6 拓扑排序

对AOV-网络的顶点进行拓扑排序，就是对各个活动排出一个线性的顺序关系。

AOV的拓扑序列如果从顶点ai到顶点aj有路径，那么在这个线性的序列中ai一定排在aj之前。

在一个有向图G中，若用顶点表示活动后任务，用边表示活动或任务之间的先后关系，则称此有向图G 为用顶点表示活动的网络，即 AOV-网络。

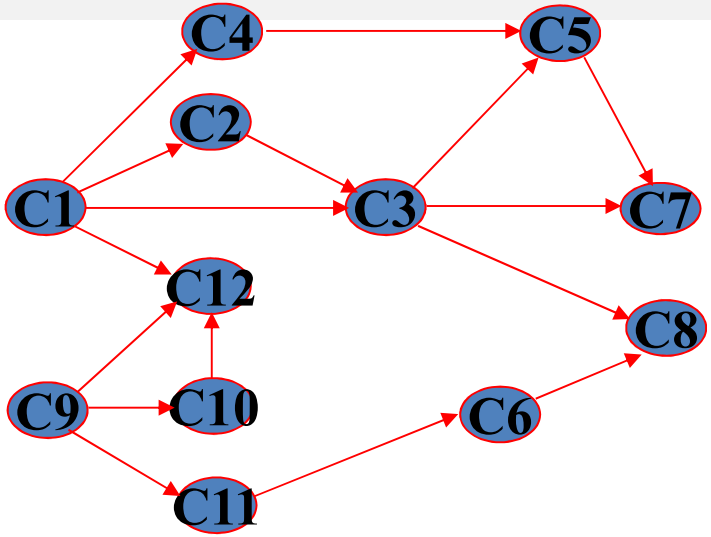
对AOV网络的的顶点进行拓扑排序——得到活动的执行顺序。

jyang
@cs.ecnu.edu.cn

6 拓扑排序

例：

课程代号	课程名称	先修课
C1	程序设计基础	无
C2	离散数学	C1
C3	数据结构	C1,C2
C4	汇编语言	C1
C5	语言的设计和分析	C3,C4
C6	计算机原理	C11
C7	编译原理	C3.C5
C8	操作系统	C3.C6



拓扑有序序列：

C1 , C2 , C3 , C4 , C5 , C7 , C9 , C10 , C11, C6 , C12 , C8 或
C9 , C10 , C11 , C6 , C1 , C12 , C4 , C2 , C3 , C5 , C7 , C8

6 拓扑排序

在AOV-网络中，如果顶点V_i的活动必须在顶点V_j的活动以前进行，则称V_i为V_j的前趋顶点，而称V_j为V_i的后继顶点。这种前趋后继关系有传递性。

AOV网络中一定不能有有向环路。如图7.6.2，V_2是V_3的前趋顶点，V_3是V_4的前趋顶点，V_4又是V_2的前趋顶点，环路表示顶点之间的先后关系进入了死循环。**因此，对给定的AOV网络首先要判定网络中是否存在环路，只有有向无环路网络在应用中才有实际意义。**

求出拓扑序列的过程，就是拓扑排序。

6 拓扑排序

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。

对AOV网进行拓扑排序的方法：

（1）在网络中选择一个没有前趋的顶点，输出。

（2）从网络中删除该顶点和以它为尾的所有有向边。

重复执行上述两个步骤，直到所有顶点都被输出，或者网络上的顶点都有前趋顶点（有回路）为止。

6 拓扑排序

实现：用邻接表作为存储结构存储有向图，头指针存放于顺序表，并且顺序表的每个结点都增加一个存放顶点入度的字段。

当某个顶点的入度为0时，就将此顶点输出，同时将该顶点的所有后继顶点的入度减1。

为了避免重复检测入度为零的顶点，可以设立一个栈，用以存放入度为零的顶点。

6 拓扑排序

算法：

（1）建立邻接表。

（2）查找邻接表中入度为零的顶点，让入度为零的顶点进栈。

（3）当栈不空时，

（a）使用出栈操作，取得栈顶的顶点j，并输出j；

（b）在邻接表第j个链表中，查找顶点j的所有后继顶点k，将顶点k的入度减1。如果顶点k的入度变为零，则顶点k进栈，转（3）

（4）当栈空时，若有向图的所有顶点都输出，则拓扑排序过程正常结束；否则，有向图存在回路。

jyang
@cs.ecnu.edu.cn

第七章　图

6 拓扑排序

P229代码段：

1. 邻接表存储
2. 实现时没有另外使用栈，借助表头结点来实现。

时间复杂度为：O（m+n）

问题：

1.某些图的拓扑序列不唯一。当两个结点之间不具有半序关系时，它们在程序输出中的先后顺序由什么决定？

2.如果在上述情况下，要求结点序号小的先输出，如何修改？

3.如何利用拓扑排序判断一个有向图中是否存在环？

jyang
@cs.ecnu.edu.cn

7 关键路径

AOV-网络：顶点表示活动或任务，边表示先后关系。
AOE-网络：在带权有向图中，顶点表示事件，**有向边表示活动，**边上的权表示完成这一活动所需要的时间，则称此有向图为用边表示活动的网络，即AOE-网络（activity on edge network）。

顶点表示的事件：以它为头顶点的边所代表的活动均已完成，以它为尾顶点的边所代表的活动可以开始这样**一种状态**。

表示实际工程的AOE-网络应该没有有向回路，存在唯一的入度为0 的开始顶点，唯一的出度为0的结束顶点。

7 关键路径

对AOE-网络，我们所关心的问题：

（1）完成整个工程至少需要多少时间？

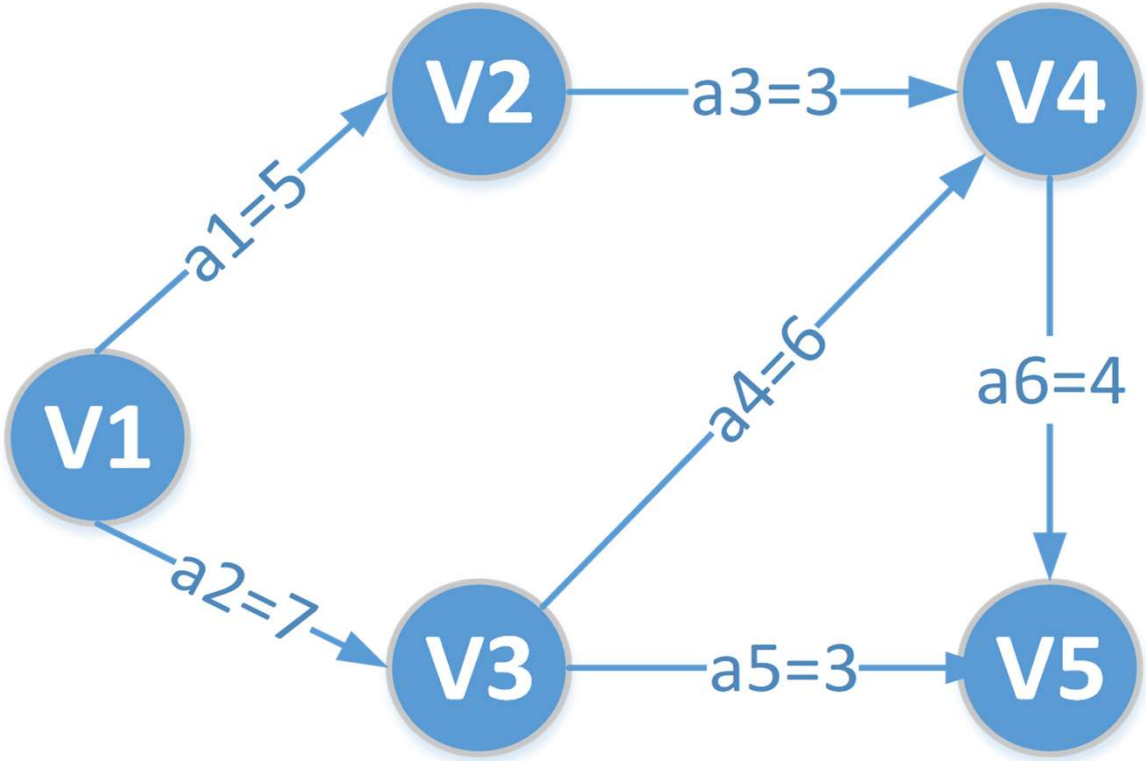
结论：完成工程的最小时间是从开始顶点到结束顶点的最长的路径长度（路径长度等于完成这条路径上各个活动所需时间之和）。

我们称从开始顶点到结束顶点的最长路径为关键路径。

（2）哪些活动是影响工程进度的关键？

一个AOE-网络可以有多条关键路径。关键路径上的所有活动是关键活动。

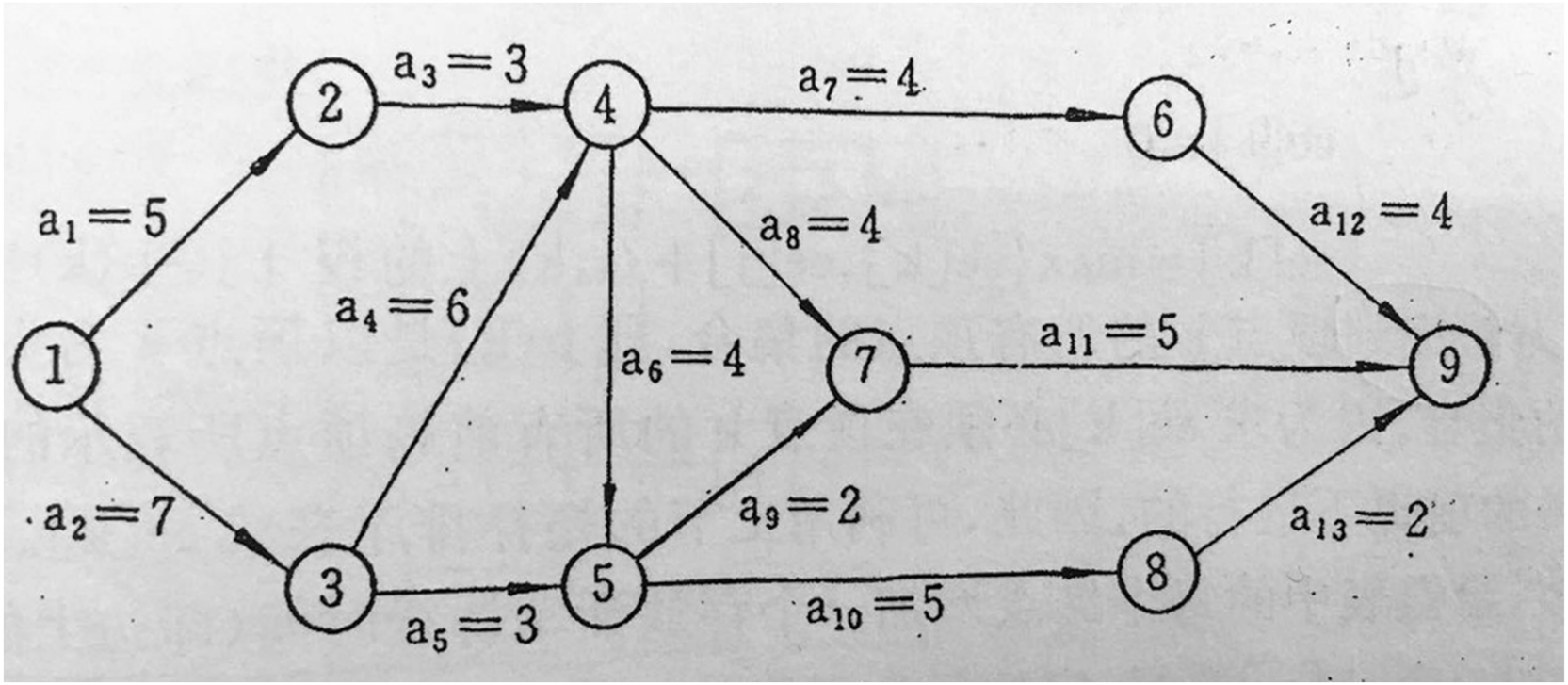
第七章　图



(1)完成整个工程至少需要多少时间？
(2)哪些活动是影响工程进度的关键？

jyang
@cs.ecnu.edu.cn

第七章　图



<1,3,4,5,7,9> 是一条关键路径，这条路径的长度是24
<1,3,4,5,8,9>也是一条关键路径，长度24

jyang
@cs.ecnu.edu.cn

7 关键路径

定义：
事件Vi（用顶点表示）能够发生的最早时间ee[i]：从开始顶点V1到顶点Vi的最长路径长度。
事件Vi允许的最迟发生时间le[i]：在保证结束顶点n在ee[n]时刻发生的前提下，事件Vi允许发生的最迟时间。
Le[i]等于ee[n]减去顶点i到顶点n的最长路径长度。

活动（用边表示）：
如果活动ai是由边<j,k>表示的，那么ai的最早开始时间e[i]=ee[j]
活动ai允许的最迟开始时间l[i]等于le[k]-(ai所需的时间)。

我们称e[i]=l[i]的活动ai为关键活动。若ai拖延时间，则整个工程也要拖延时间。
L[i]-e[i]为余量， L[i]-e[i]>0的活动不是关键活动。

7 关键路径

求关键路径：

1）对每个顶点，求 ee[i], le[i]
2）对每条边（也就是每个活动a_i），求 e[i]和l[i]
3) 如果e[i]=l[i]，则这条边属于关键路径。

7 关键路径

求ee[k]

利用拓扑排序的程序来求（因为求ee[k]必须在顶点k的所有前趋顶点所表示的事件的最早发生时间都已求得的前提下进行）。

$$\begin{cases} ee[1] = 0 \\ ee[k] = max\{ee[k], ee[j] + < j,k > \text{上的权} \mid j \in p(k)\} \end{cases}$$

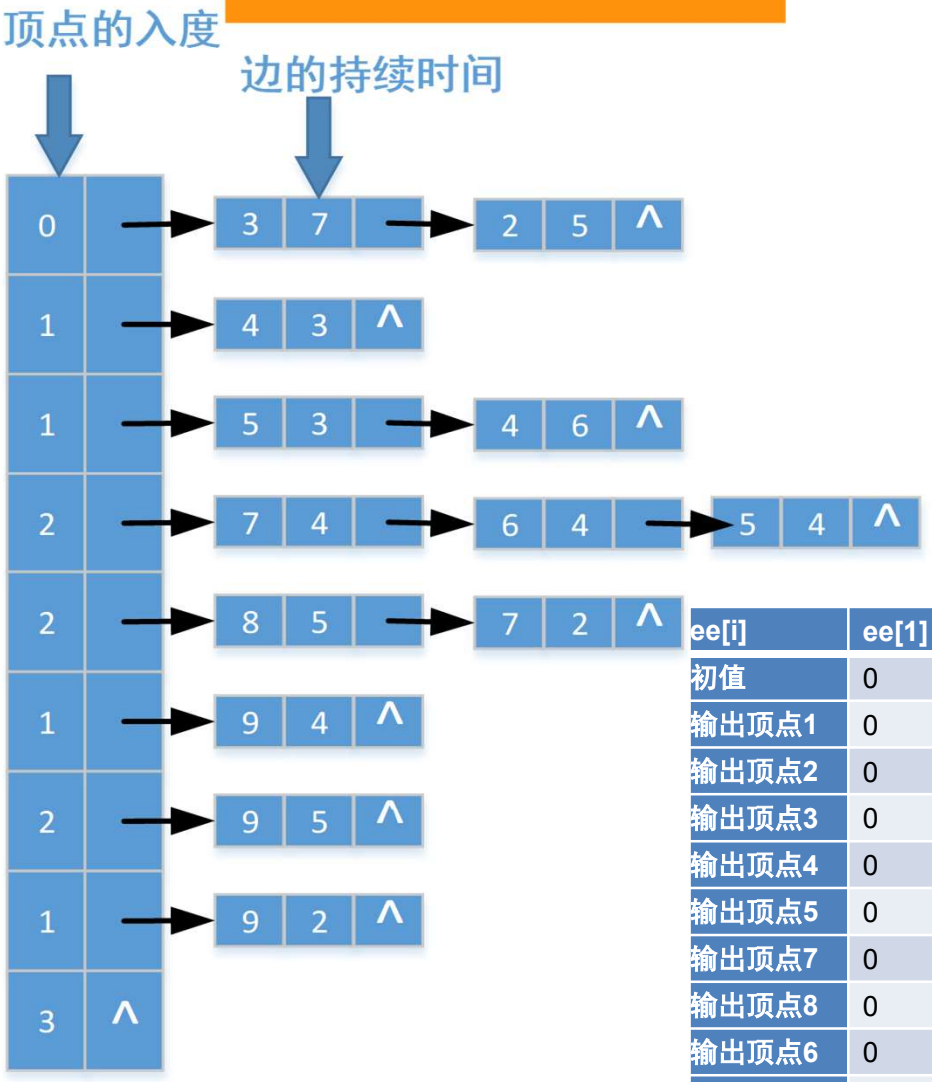
其中，p(k)为邻接到顶点k的所有顶点的集合，即p(k)是以顶点k为头顶点的所有 有向边 的尾顶点的集合。

7 关键路径

```
//ee[n]赋初值0
while(top!=0)  //栈不空
{           j=top;       //j 是出栈的顶点
            top=ch[top].count;       //出栈
            tpv[++i]=j;
            t=ch[j].head;
            while(t!=NULL)      //相关顶点的入度减一，即“拿走”以j为尾的边
             {          k=t->ver;
                        if(--(ch[k].count)==0) {    //减一，如果减后入度为0，入栈
                                    ch[k].count=top;            top=k;}
                        if (ee[k]<ee[j]+t->dur)  ee[k]= ee[j]+t->dur;
//”拿走”<j,k>这条边后，计算由这条边到达ee[k]的路径长度。
                        t=t->link;}
            }
}
```

jyang
@cs.ecnu.edu.cn

第七章　图



ee[i]	ee[1]	ee [2]	ee [3]	ee [4]	ee [5]	ee [6]	ee [7]	ee [8]	ee [9]	栈
初值	0	0	0	0	0	0	0	0	0	1
输出顶点1	0	5	7	0	0	0	0	0	0	3,2
输出顶点2	0	5	7	8	0	0	0	0	0	3
输出顶点3	0	5	7	13	0	0	0	0	0	4
输出顶点4	0	5	7	13	17	17	17	0	0	6,5
输出顶点5	0	5	7	13	17	17	19	22	0	6,8,7
输出顶点7	0	5	7	13	17	17	19	22	24	6,8
输出顶点8	0	5	7	13	17	17	19	22	24	6
输出顶点6	0	5	7	13	17	17	19	22	24	9
输出顶点9	0	5	7	13	17	17	19	22	24	

得到的拓扑序列：1,2,3,4,5,7,8,6,9

jyang
@cs.ecnu.edu.cn

第七章　图

ee[i]	ee[1]	ee [2]	ee [3]	ee [4]	ee [5]	ee [6]	ee [7]	ee [8]	ee [9]	栈
初值	0	0	0	0	0	0	0	0	0	1
输出顶点1	0	5	7	0	0	0	0	0	0	3,2
输出顶点2	0	5	7	8	0	0	0	0	0	3
输出顶点3	0	5	7	13	0	0	0	0	0	4
输出顶点4	0	5	7	13	17	17	17	0	0	6,5
输出顶点5	0	5	7	13	17	17	19	22	0	6,8,7
输出顶点7	0	5	7	13	17	17	19	22	24	6,8
输出顶点8	0	5	7	13	17	17	19	22	24	6
输出顶点6	0	5	7	13	17	17	19	22	24	9
输出顶点9	0	5	7	13	17	17	19	22	24	

7 关键路径

求le[k] 从后往前。

$$\begin{cases} le[n] = ee[n] \\ le[k] = min\{le[k], le[j] - < k, j > \text{上的权} \mid j \in s(k)\} \end{cases}$$

S(k)是邻接于顶点k的所有顶点的集合。

计算方法：

首先求出ee，并同时得到顶点的拓扑序列。把序列倒排，就得到顶点的逆拓扑序列。然后用上述公式计算le。

jyang
@cs.ecnu.edu.cn

第七章　图

拓扑序列：1,2,3,4,5,7,8,6,9
逆序：9,6,8,7,5,4,3,2,1

le[6]=le[9]-<6,9>的权,20

le[8]=le[9]-<8,9>的权,22

le[7]=le[9]-<7,9>的权,19

le[5] min { le[8]-<5,8>的权， le[7]-<5,7
min（17，17）

le[4] min（19-4，20-4,17-4）

le[3] min（17-3，,13-6）

le[2] min（13-3）

le[1] min（7-7,10-5）

le[i]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
	0	10	7	13	17	20	19	22	24

jyang
@cs.ecnu.edu.cn

7 关键路径

在求出AOE-网络中所有顶点的最早发生时间ee和最迟发生时间le后，如果活动a_i用边<j,k>表示，则可以计算活动的最早开始时间和最迟开始时间。

$$\begin{cases} e[i] = ee[j] \\ l[i] = le[k] - <j,k>\text{上的权} \end{cases}$$

活动的最早开始时间 等于尾顶点的最早开始时间。

活动的最晚开始时间，等于头顶点的最晚开始时间减去边的权

我们称e[i]=l[i]的活动为关键活动。

删去所有非关键活动，就到带权有向图。图中所有路径都是关键路径。

第七章　图

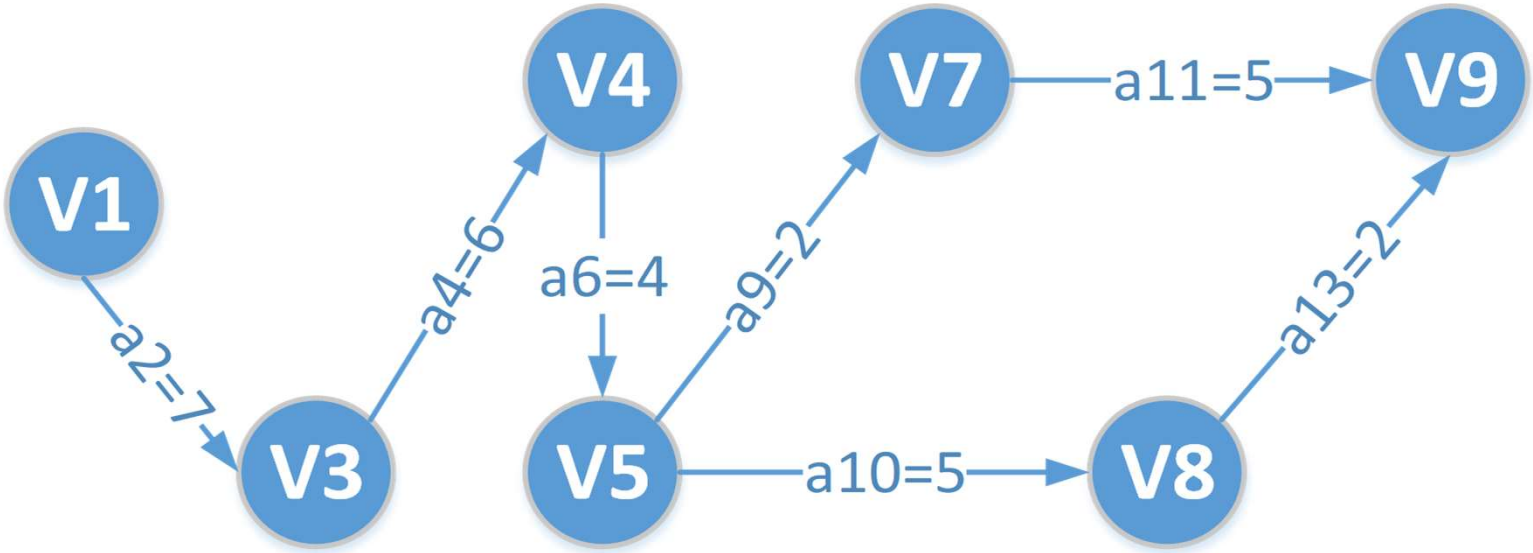
ee[1]	ee [2]	ee [3]	ee [4]	ee [5]	ee [6]	ee [7]	ee [8]	ee [9]
0	5	7	13	17	17	19	22	24

le[1]	le [2]	le [3]	le [4]	le [5]	le [6]	le [7]	le [8]	le [9]
0	10	7	13	17	20	19	22	24

活动		e[i] 尾顶点的最早开始时间	L[i] 头顶点的最晚开始时间减去边上的时间
a1	<1,2>5	0	5
a2	<1,3>7	0	0
a3	<2,4>3	5	10
a4	<3,4>6	7	7
a5	<3,5>3	7	14
a6	<4,5>4	13	13
a7	<4,6>4	13	16
a8	<4,7>4	13	15
a9	<5,7>2	17	17
a10	<5,8>5	17	17
a11	<7,9>5	19	19
a12	<6,9>4	17	20
a13	<8,9>2	22	22

jyang
@cs.ecnu.edu.cn

第七章　图

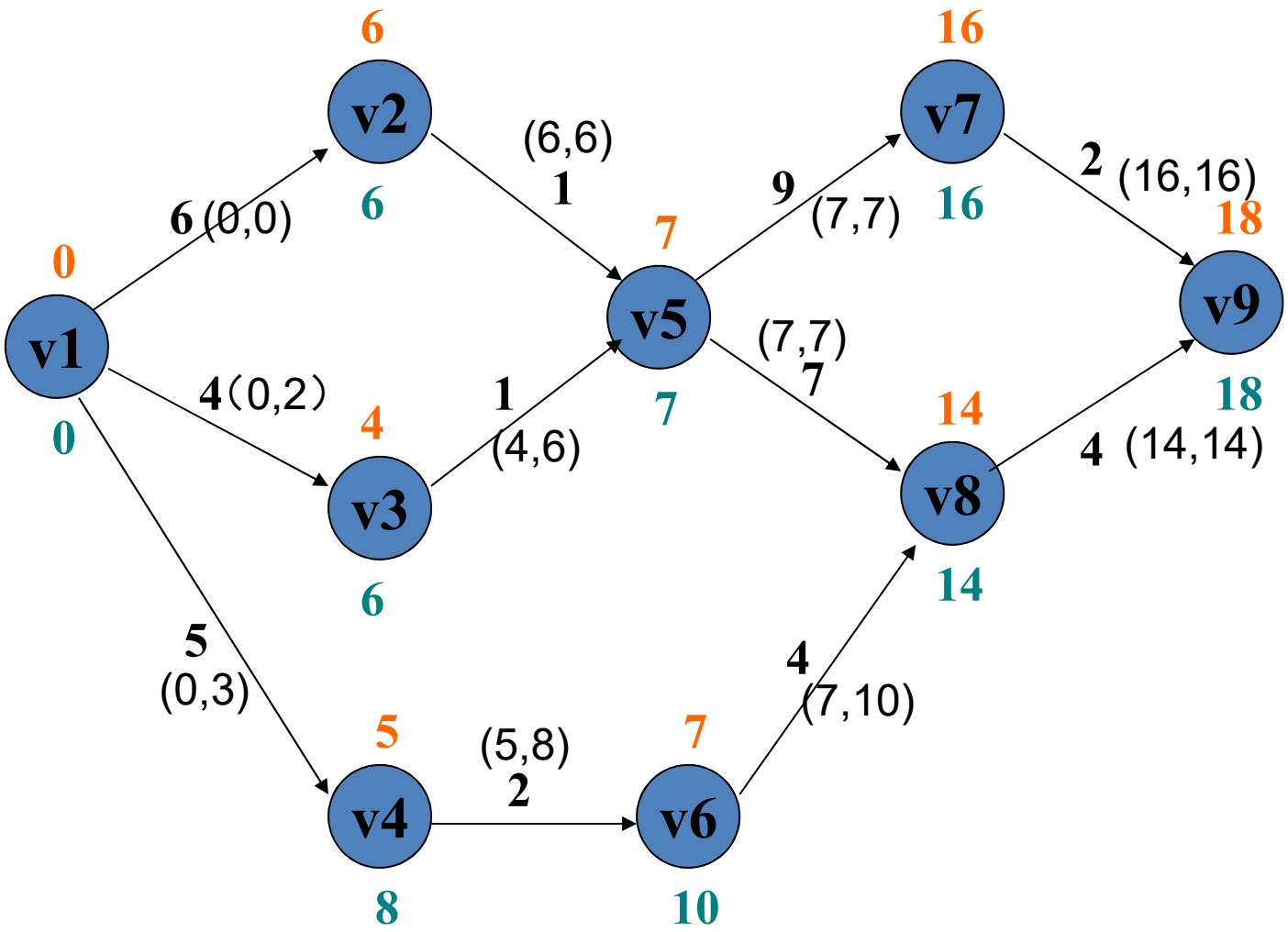


关键路径：决定完成时间。

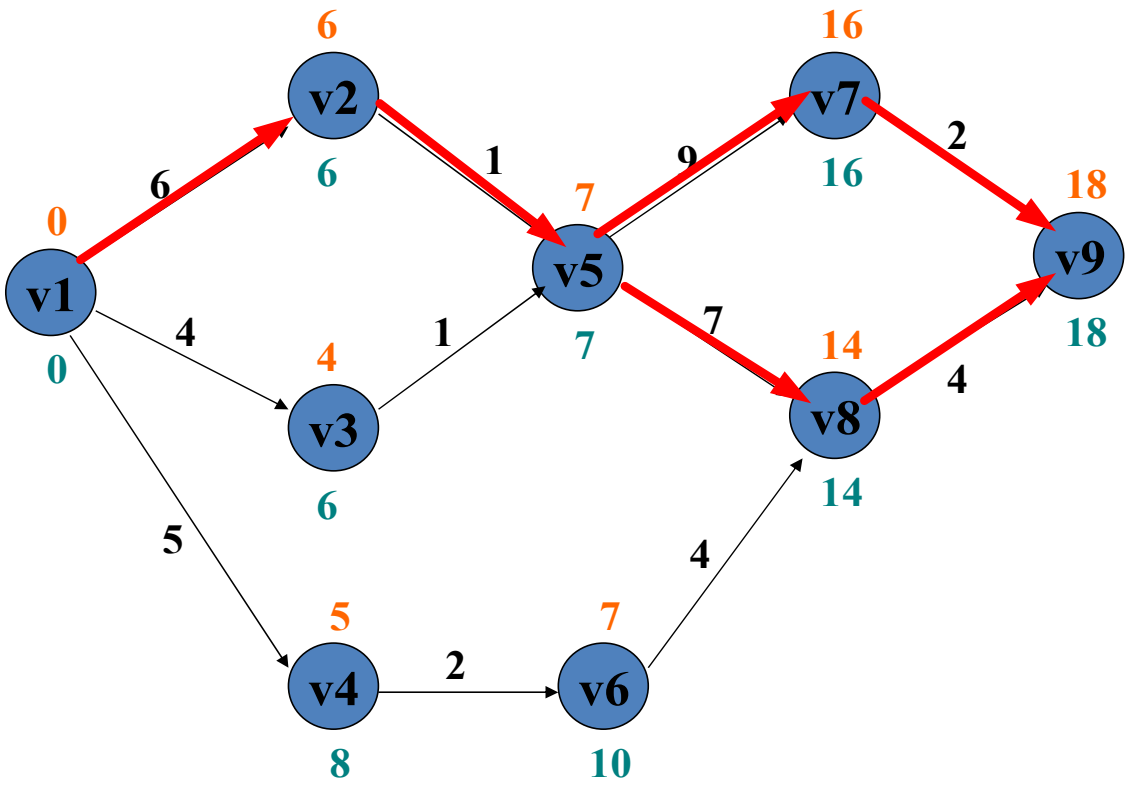
并不是加快任何一个关键活动都可以缩短整个工程的完成时间，只有加快那些包含在所有的关键路径上的关键活动才能达到这个目的。

但一旦改变了某个路径（即活动）的时间，就需要重新计算关键路径。

7 关键路径

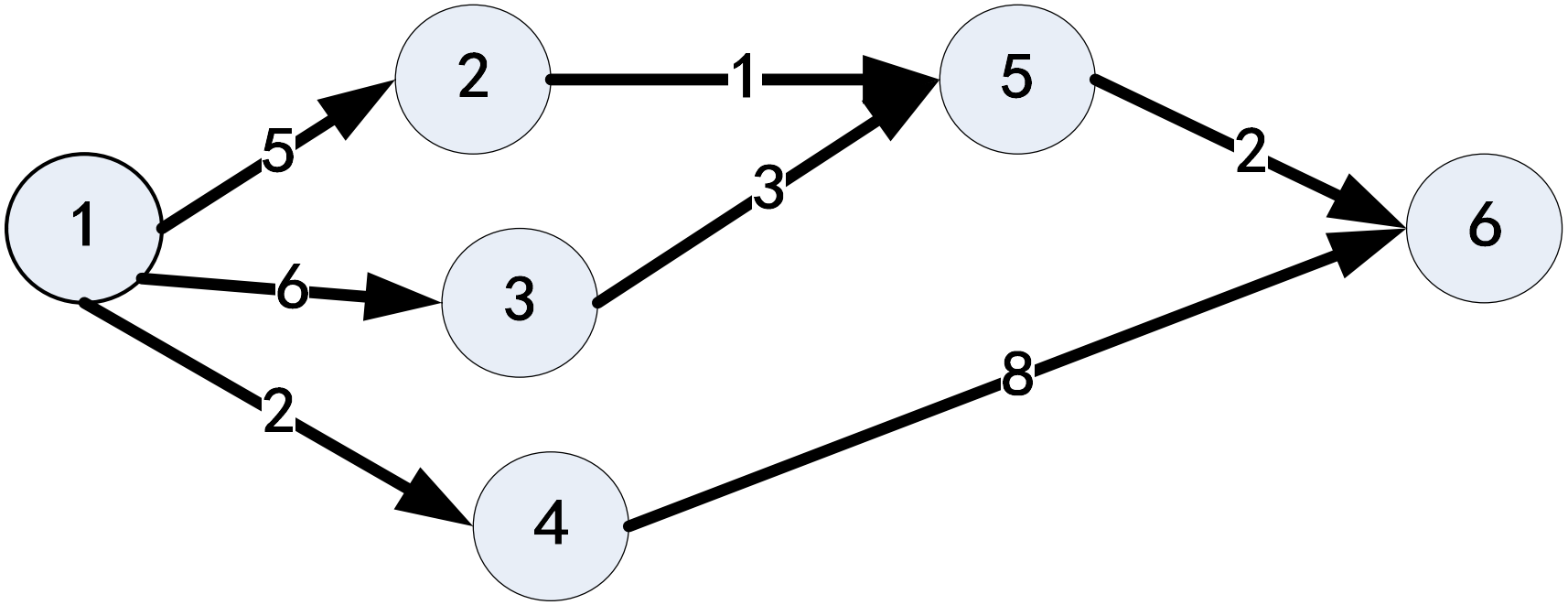

v1
v2
v3
v4
v5
v6
v7
v8
v9
0
6
4
5
7
7
16
14
18
0
6
6
8
7
16
14
18
6 (0,0)
4 (0,2)
5 (0,3)
(6,6) 1
1 (4,6)
(5,8) 2
9 (7,7)
(7,7) 7
4 (7,10)
2 (16,16)
4 (14,14)

第七章　图


v1
v2
v3
v4
v5
v6
v7
v8
v9
0
0
6
6
4
6
7
7
5
8
16
16
14
14
18
18
6
4
5
1
1
2
9
7
4
2
4

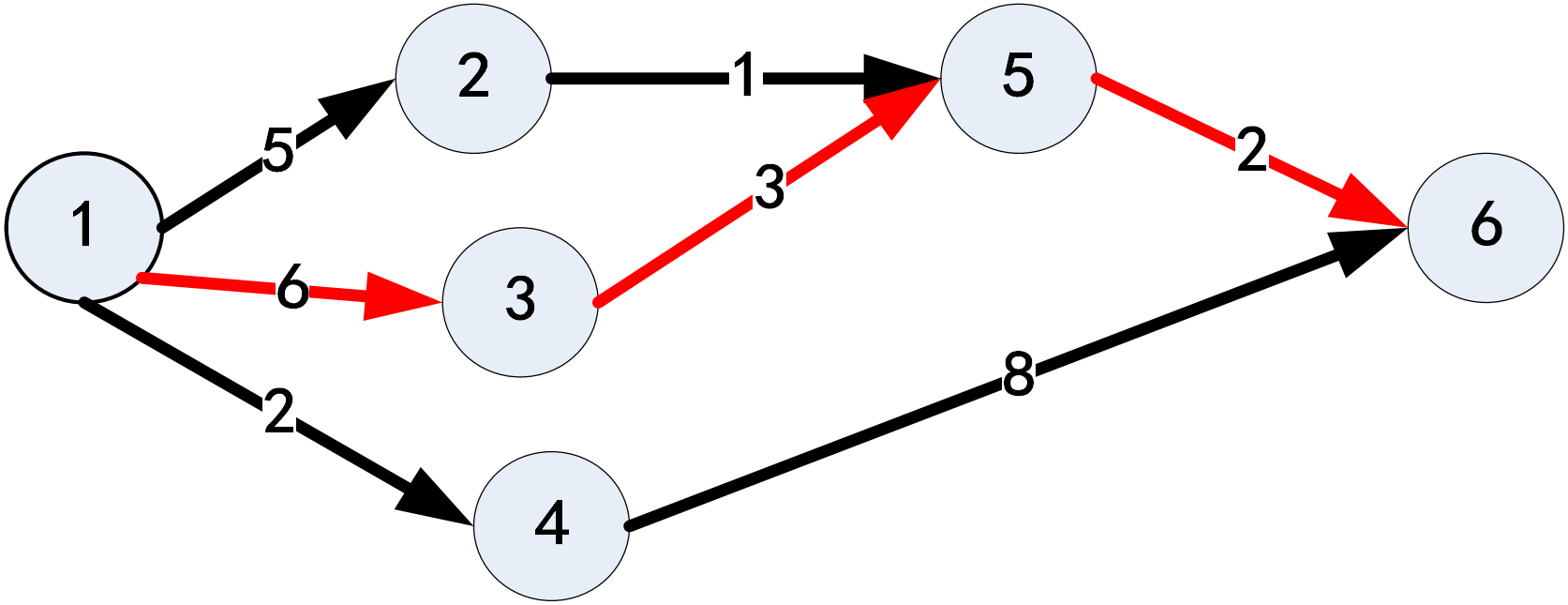
7 关键路径

例：



7 关键路径

例：



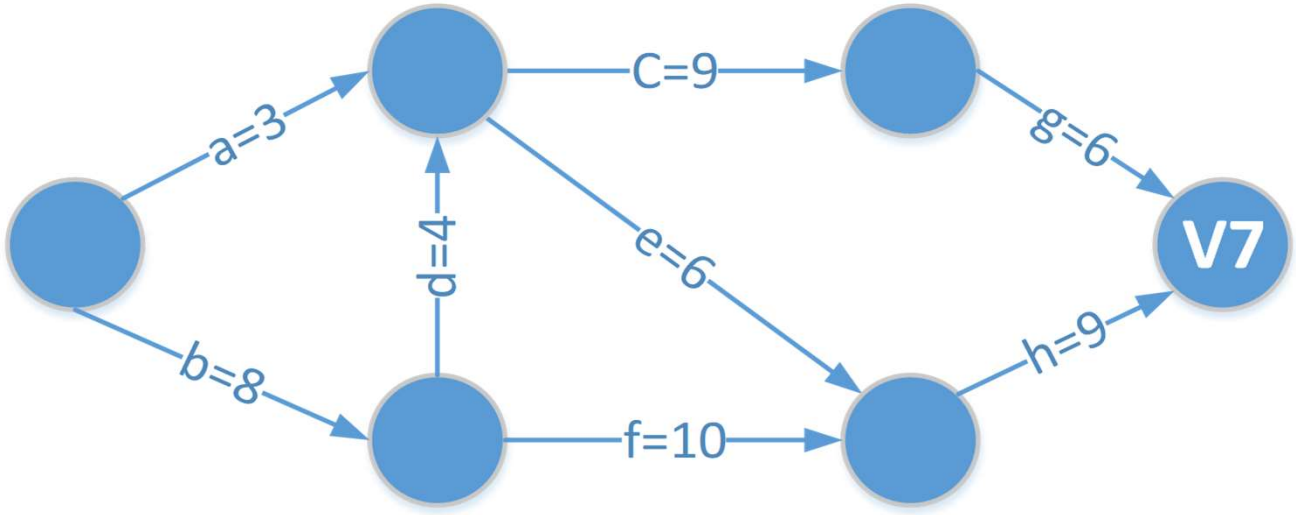
jyang
@cs.ecnu.edu.cn

第七章　图

练习：

下图的AOE网表示一项包含8个活动的工程。通过同时加快若干活动的进度可以缩短整个工程的工期。下列选项中，加快其进度就可以缩短工程工期的是（　）

A. c和e　　B. d和e　　C. f和d　　D. f和h



jyang
@cs.ecnu.edu.cn

第七章　图

基本概念
存储结构：邻接矩阵，邻接表
深度优先遍历，广度优先遍历，连通分量

无向图：生成树
无向图，边上带权：最小代价生成树 （“权”和最小 ）Prim，Kruskal

有向图：拓扑排序（AOV网）
有向图，边上带权：
最短路径（“权”和最小）Dijkstra，Floyd
关键路径（AOE网） 拓扑+“最长”